



PPG INSTITUTE OF TECHNOLOGY

(Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai)
Recognized by UGC | Accredited by NAAC with A | ISO Certified Institution
NH 209, Sathy Main Road, Saravanampatti, Coimbatore – 641035, Tamil Nadu



CCS342 – DEVOPS LABORATORY RECORD

**DEPARTMENT OF
INFORMATION TECHNOLOGY**



PPG INSTITUTE OF TECHNOLOGY

(Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai)
Recognized by UGC | Accredited by NAAC with A | ISO Certified Institution
NH 209, Sathy Main Road, Saravanampatti, Coimbatore – 641035, Tamil Nadu



DEPARTMENT OF INFORMATION TECHNOLOGY

CCS342 - DEVOPS LABORATORY RECORD

NAME:..... ROLL NO:.....

SEMESTER: BRANCH:

Certified bonafide record of work done by.....

PLACE: COIMBATORE. DATE:.....

Staff In-Charge

Head of the Department

University Register Number:.....

Submitted for the University Practical Examination held on.....

INTERNAL EXAMINER

EXTERNAL EXAMINER

TABLE OF CONTENTS

Exp. No.	Date	Title of the Experiment	Page No.	Marks	Signature
1.		Create Maven Build Pipeline in Azure	4		
2.		Run Regression Tests Using Maven Build Pipeline in Azure	12		
3.		Install Jenkins in Cloud	20		
4.		Create CI Pipeline using Jenkins	37		
5.		Create CD Pipeline using Jenkins And Deploy In Cloud	55		
6.		Create an Ansible Playbook for a Simple Web Application Infrastructure	58		
7.		Build A Simple Application Using Gradle	60		
8.		Install Ansible and Configure Ansible Roles And to Write Playbooks	65		

TOTAL MARKS :

FACULTY INCHARGE :

Ex. No: 1

CREATE MAVEN BUILD PIPELINE IN AZURE

Date:

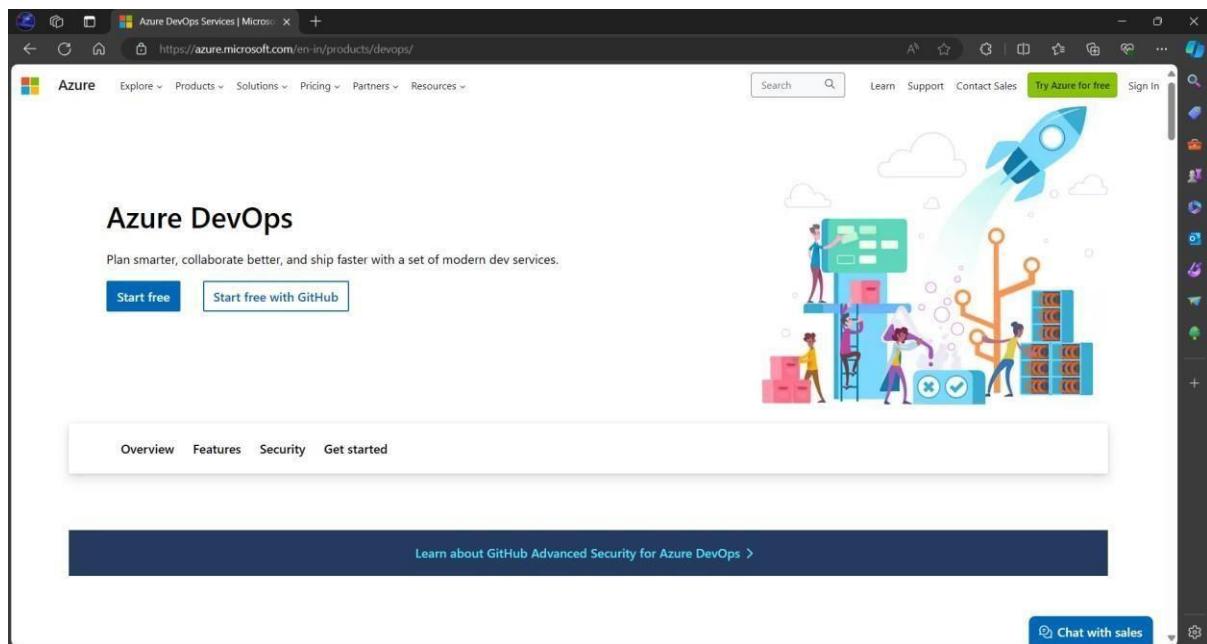
AIM

To build a maven build pipeline in Azure.

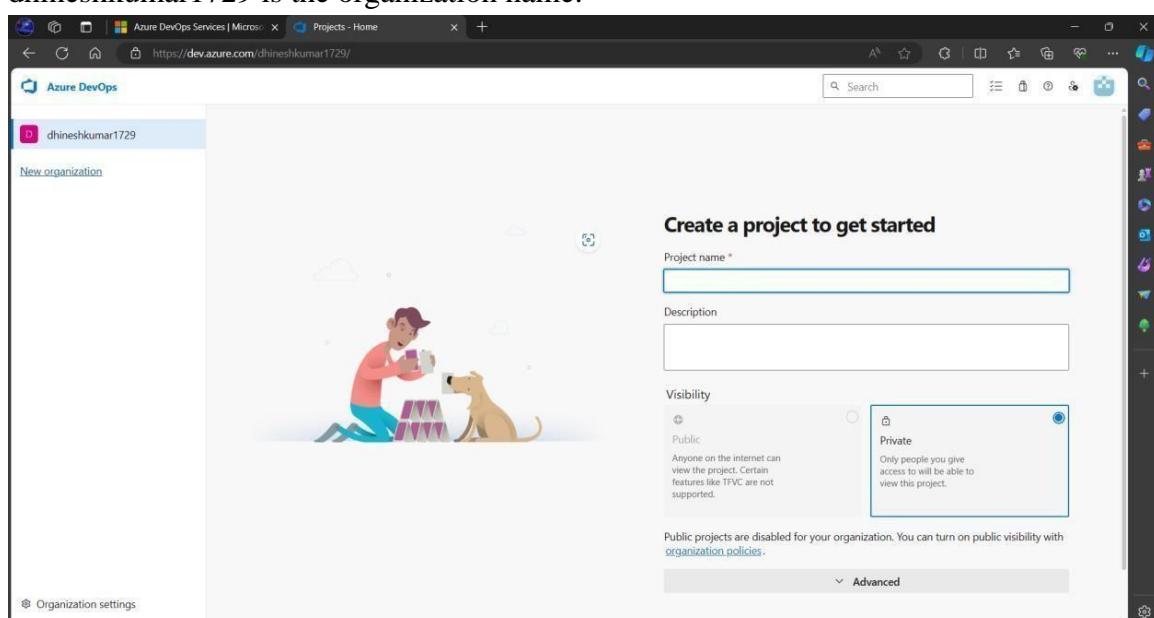
PROCEDURE

STEP 1: Create an account on the Azure DevOps website.

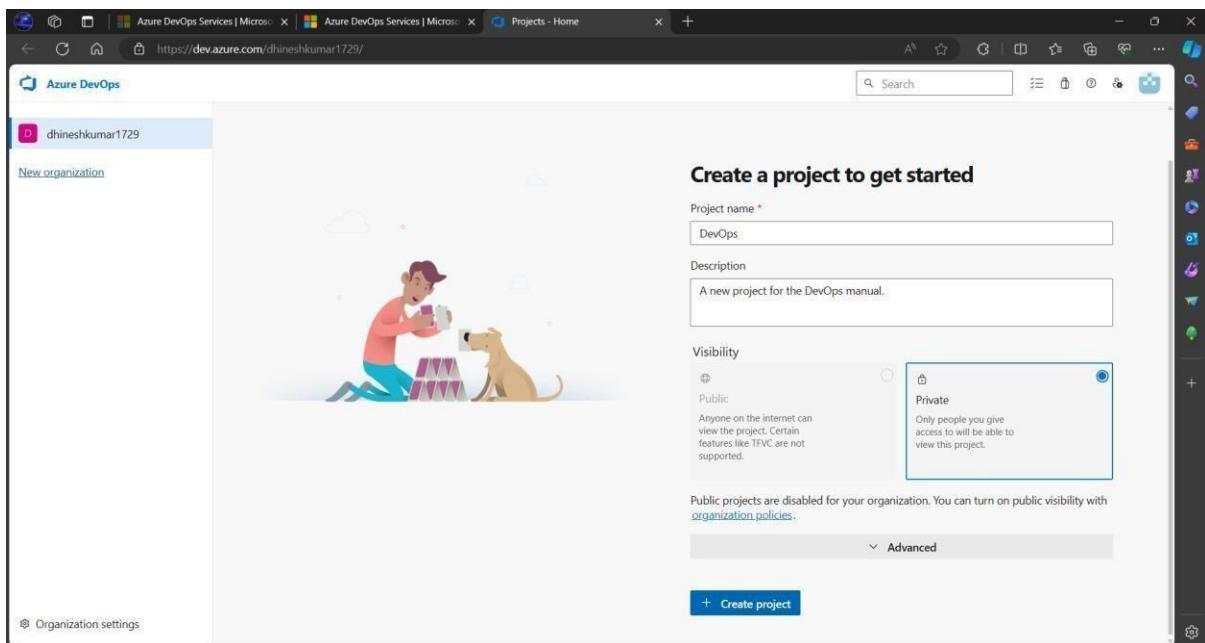
1. Go to <https://azure.microsoft.com/en-in/products/devops/> website and sign in using a GitHub account.



2. Create a new organization with a suitable name of your choice. Here dhineshkumar1729 is the organization name.



3. Create a new project named **DevOps** in private mode.



STEP 2: Installing IntelliJ IDEA

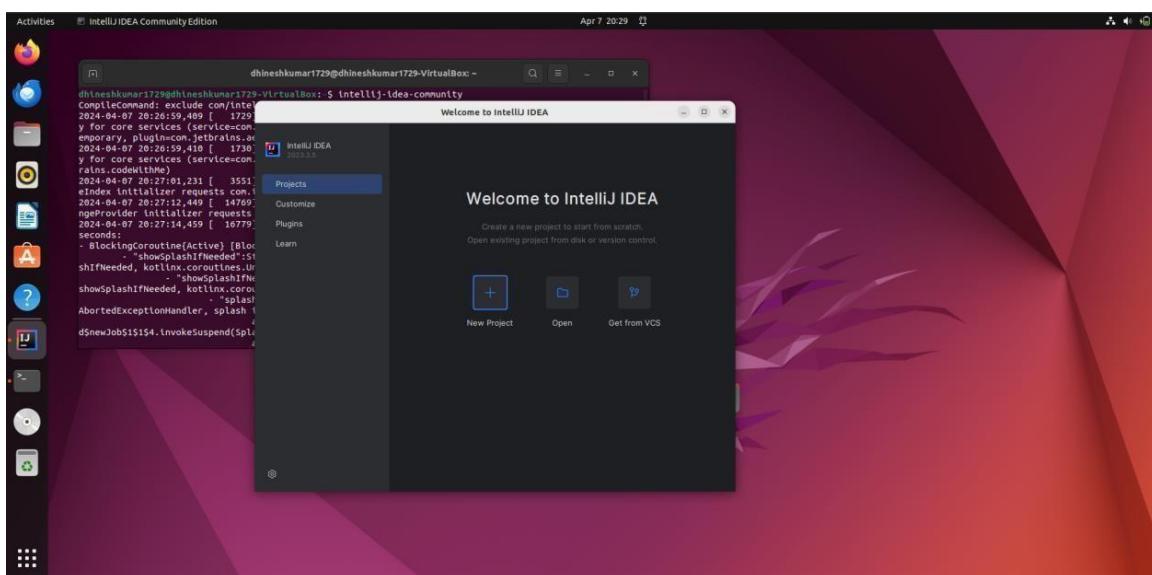
1. Download IntelliJ Idea from <https://www.jetbrains.com/idea/download/?section=linux> and activate a free trial for 30 days or through the following command.

```
$ sudo snap install intellij-idea-community --community
```

2. Open the IDE using the below command in the terminal.

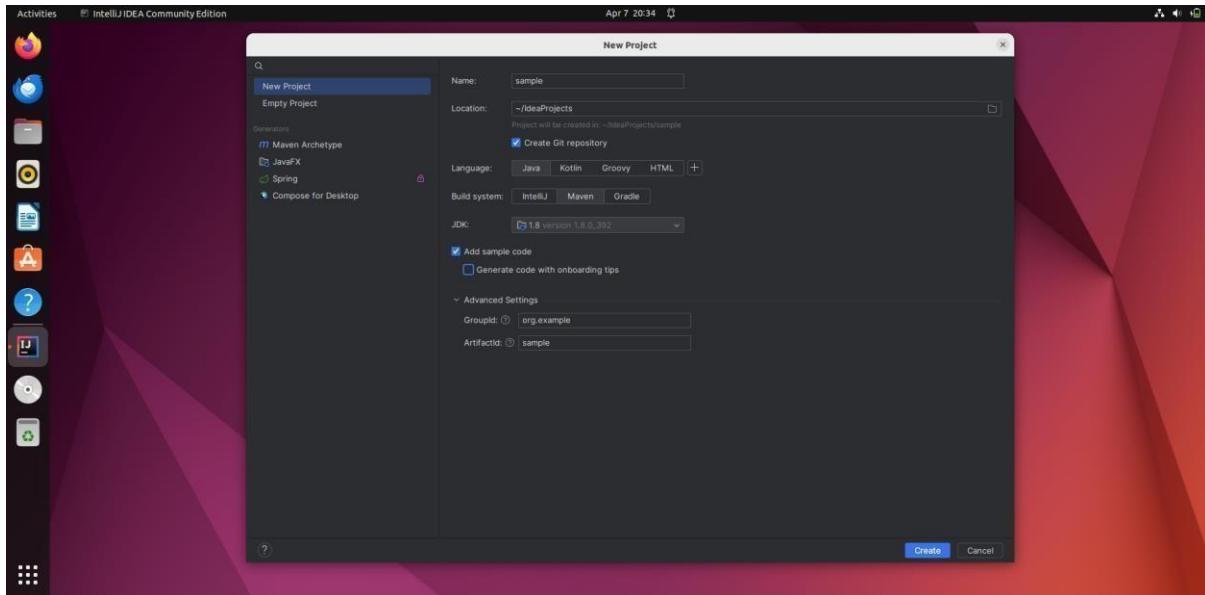
```
$ intellij-idea-community
```

3. Java is a pre-requisite for using this IDE



STEP 3: Create a new Maven project using the IDE.

1. Create a project by clicking on the New Project option, choose an appropriate name and select the following features as mentioned in the image (Build system: Maven and Language: Java)



1. A sample code has been generated. Now press ALT + F12 to open a terminal inside the IDE.

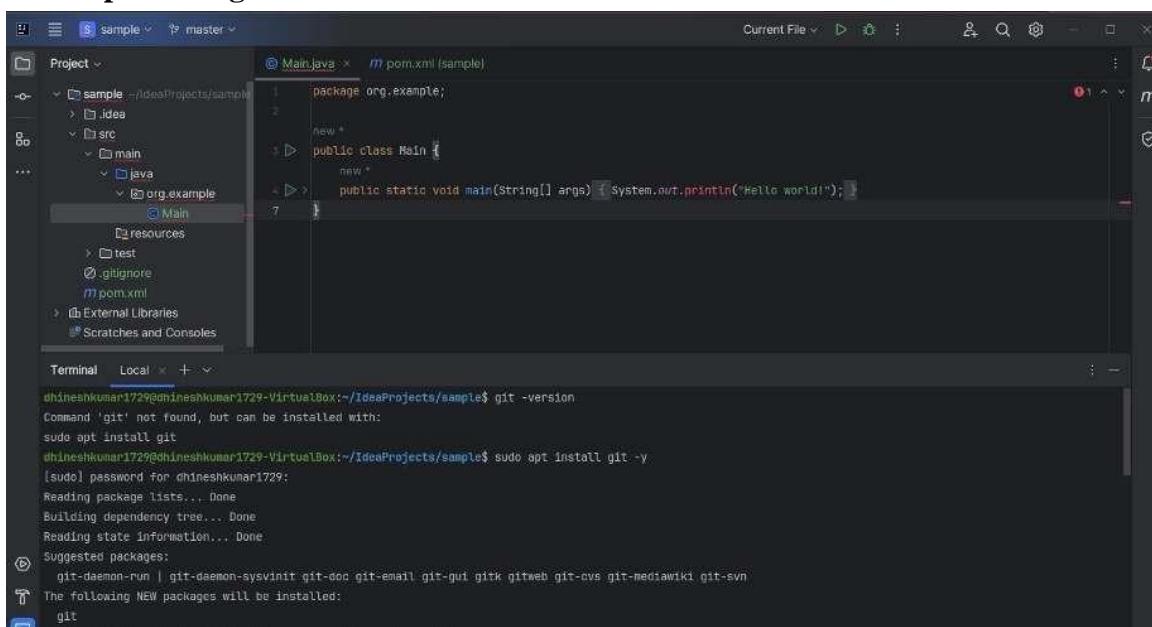
STEP 4: Install GIT and configure it

1. Check whether git is installed using the following command.

```
$ git -version
```

2. Install git using

```
$ sudo apt install git
```



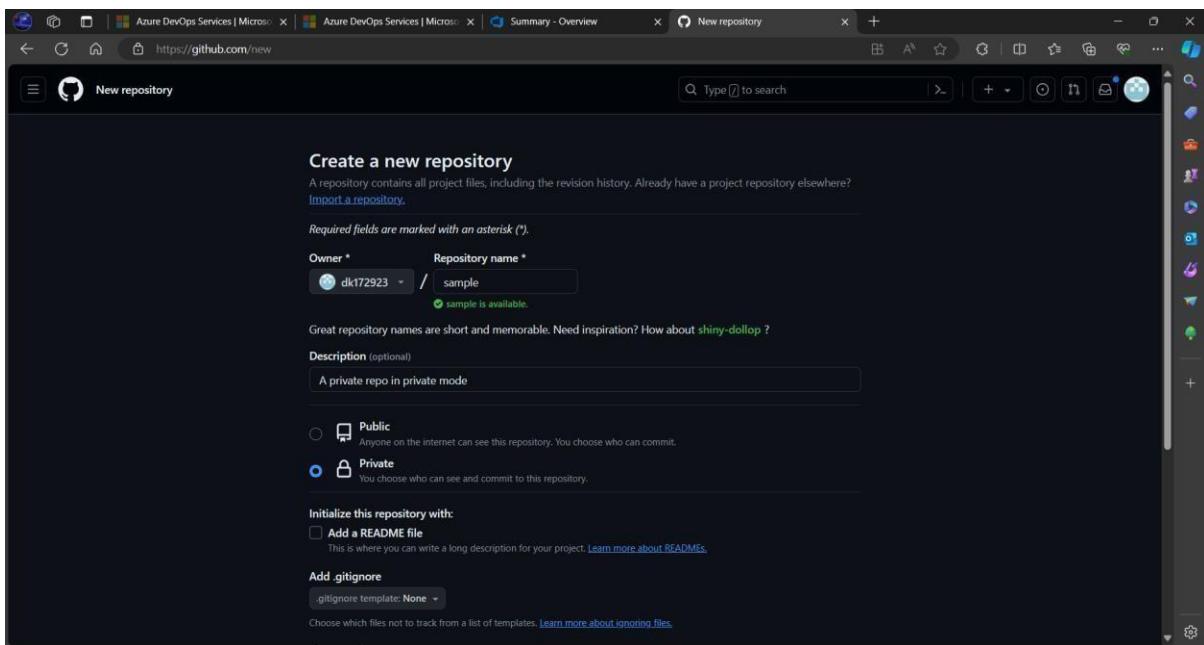
3. Once git has been installed execute the following commands one by one.

```
$ git init  
$ git add .  
$ git commit -m "First Commit"  
$ git branch -M main
```

1. Configure git using the below commands

```
$ git config --global user.name "Your_Name"  
$ git config --global user.email "Your_Email_ID"
```

1. Go to <https://github.com> sign in with your account and create a new private repository with the same name as your maven project.



1. Copy the **SSH URL** for the created repository.

2. Now go back to IDE's terminal and execute the following commands

```
$ ssh-keygen -t rsa -b 4096 -C dhineshkumar24murugan007@gmail.com
```

(press ENTER for all questions)

```
$ cat ~/.ssh/id_rsa.pub (copy the printed SSH key)
```

1. Now go to the **SSH and GPG** section in the GitHub settings option.

2. Click on the **New SSH key** button.

3. Choose a suitable name and paste the SSH key into the provided space.

4. Now get back to the IDE's terminal and type the following command

```
$ git remote set-url origin git@github.com:dk172923/sample.git
```

(Here, <git@github.com:dk172923/sample.git> is the remote URL to use SSH dk172923 – GitHub ID sample.git – repository name)

1. Now finally run the push commands to push the code to the remote repository from the local repository.

```
$ git push --set-upstream origin main (Type YES for prompted question)
```

```
$ git push
```

The screenshot shows the IntelliJ IDEA interface. On the left is the Project tool window displaying a file structure for a Java project named 'sample'. The 'Main.java' file is open in the main editor, containing the following code:

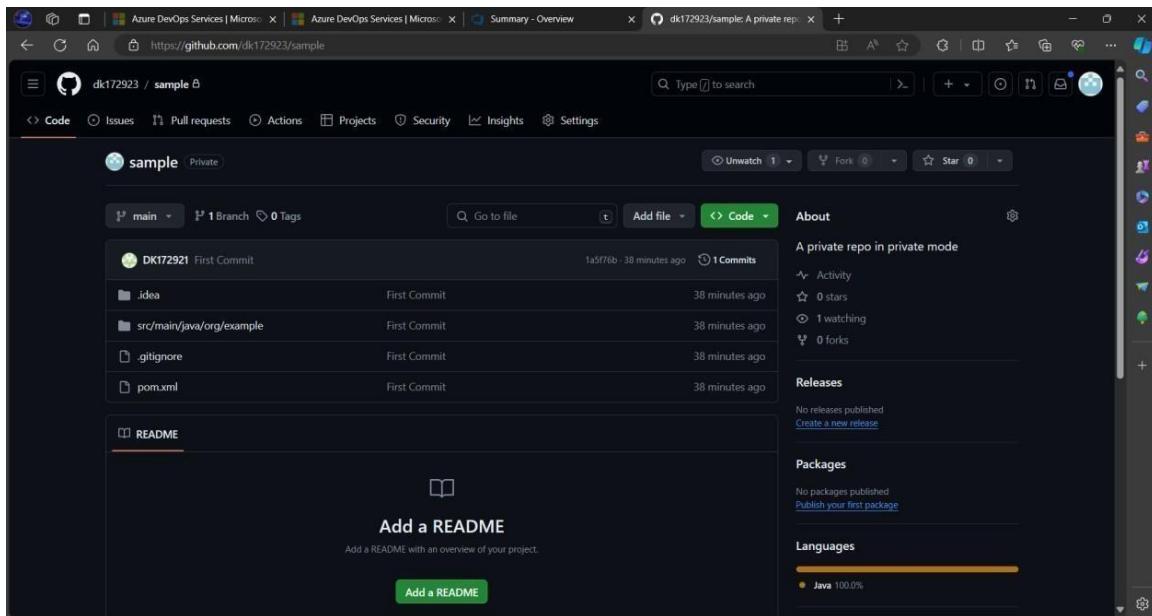
```
1 package org.example;
2
3 new *
4 public class Main {
5     new *
6     public static void main(String[] args) { System.out.println("Hello world!"); }
7 }
```

Below the editor is a terminal window showing the command-line output of a 'git push' operation:

```
dhineshkumar1729@dhineshkumar1729-VirtualBox:/IdeaProjects/sample$ git push --set-upstream origin main
The authenticity of host 'github.com (20.207.73.82)' can't be established.
ED25519 key fingerprint is SHA256:+DiY3vvvVGtUJjhbpZisF/zLDA0zPMsvHdkn4UvC0qU.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'github.com' (ED25519) to the list of known hosts.
Enumerating objects: 14, done.
Counting objects: 100% (14/14), done.
Delta compression using up to 4 threads
Compressing objects: 100% (9/9), done.
Writing objects: 100% (14/14), 2.55 KiB | 522.00 KiB/s, done.
Total 14 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:dk172923/sample.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
dhineshkumar1729@dhineshkumar1729-VirtualBox:/IdeaProjects/sample$ git push
```

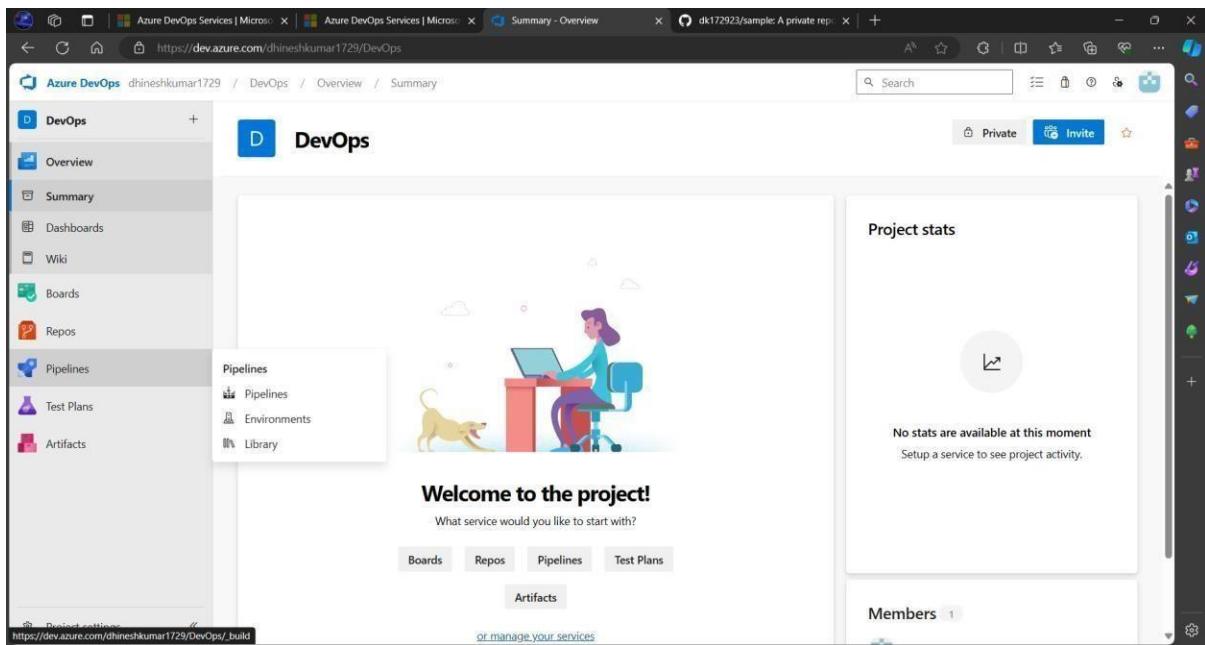
The terminal also shows the current file path: sample > src > main > java > org > example > Main.

The code has been successfully pushed to the remote repository.



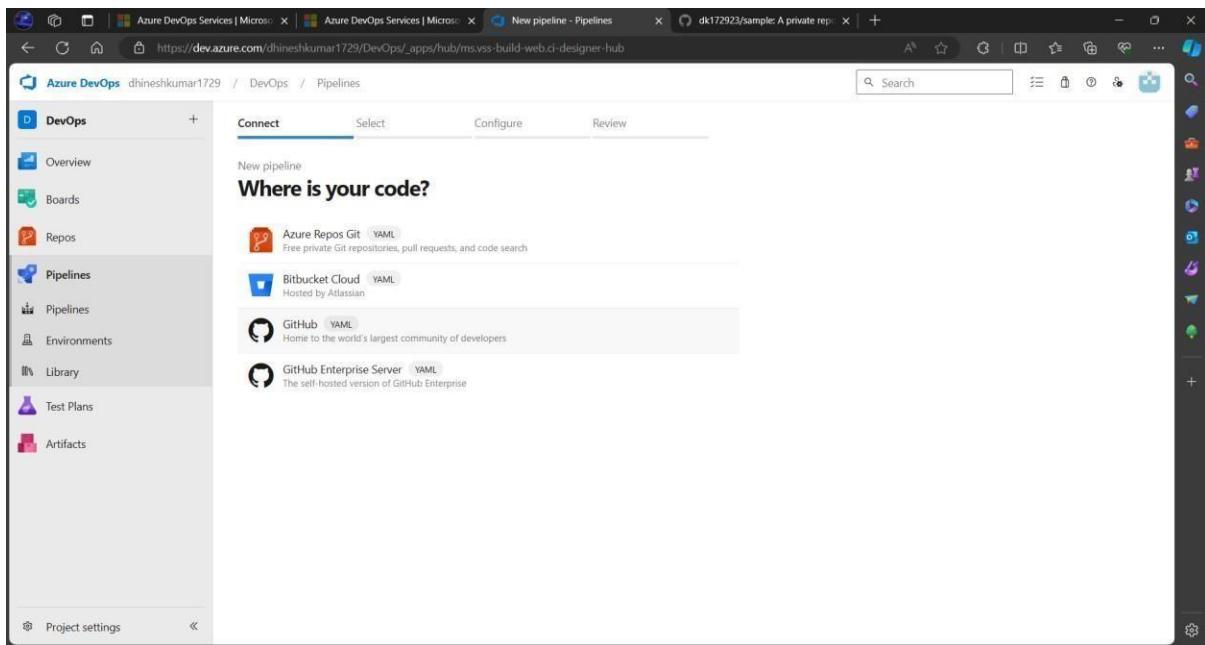
STEP 5: Create a pipeline for the created maven repository.

1. Click on Pipelines on the left pane of the Azure DevOps website.

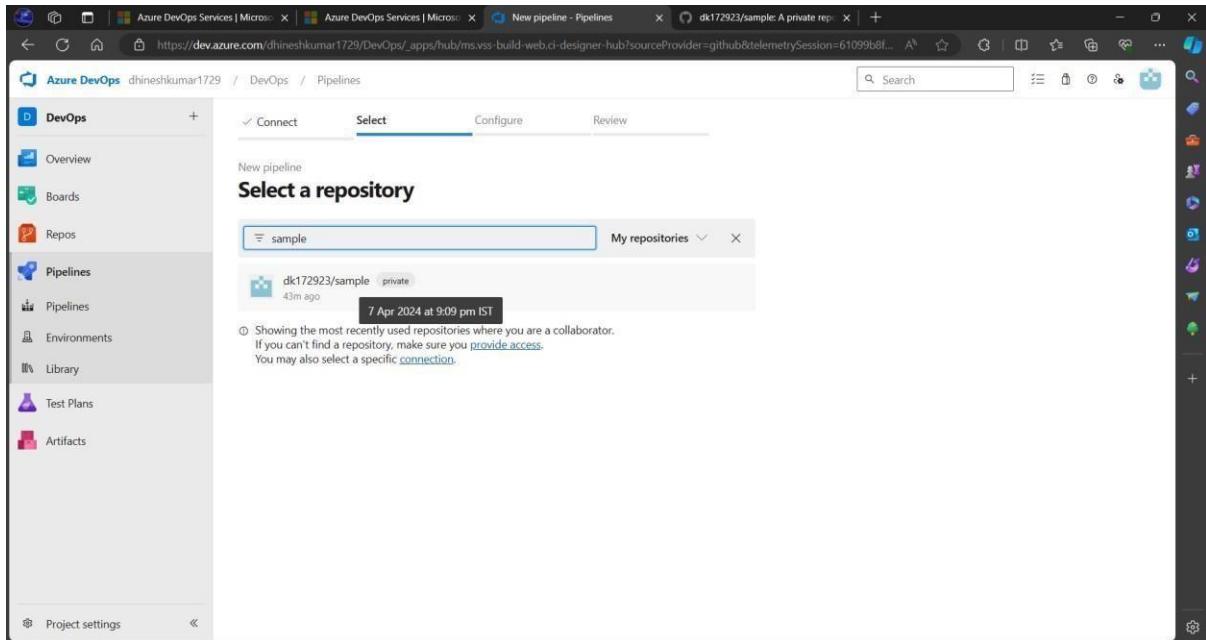


1. Click on the **Create Pipeline** button.

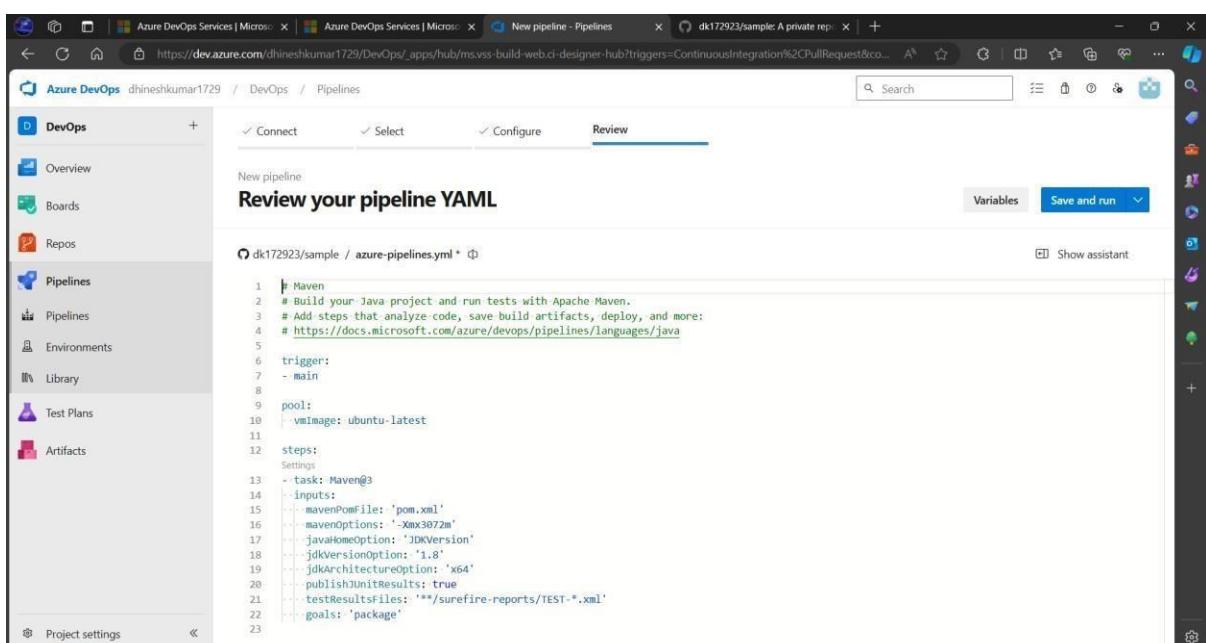
2. Choose the GitHub YAML option.



1. Select the created repository “sample” from the available repositories.
2. Give permissions if prompted to do so and sign in to your GitHub account if needed.



3. Choose Maven Pipeline from the given options.
4. A YAML file is automatically created based on the configuration details found in the pom.xml file in the repository.



1. Press the **Save and Run** button.
2. Create a commit message and press the run button again.
3. Click on the created Job from the Jobs table.

A screenshot of the Azure DevOps Pipelines interface. The pipeline name is "#20240407.1 • Set up CI with Azure Pipelines". The status is green with a checkmark, indicating success. A message says "This run is being retained as one of 3 recent runs by pipeline." There are tabs for "Summary" and "Code Coverage". Under "Summary", it shows the trigger was "dk172923", repository "dk172923/sample", commit "main 32529ff", start time "Today at 9:59 pm", duration "20s", and artifacts "0 artifacts". The "Jobs" section shows a single job named "Job" with status "Success" and duration "13s".

A successful job completion would be like this.

A screenshot of the Azure DevOps Pipelines interface showing the logs for run #20240407.1. The log title is "Finalize Job". The log content shows the following steps:
1 Starting: Finalize Job
2 Cleaning up task key
3 Start cleaning up orphan processes.
4 Finishing: Finalize Job

RESULT

Thus, a maven build pipeline has been created using Azure.

Ex. No: 2

RUN REGRESSION TESTS USING MAVEN BUILD PIPELINE IN AZURE

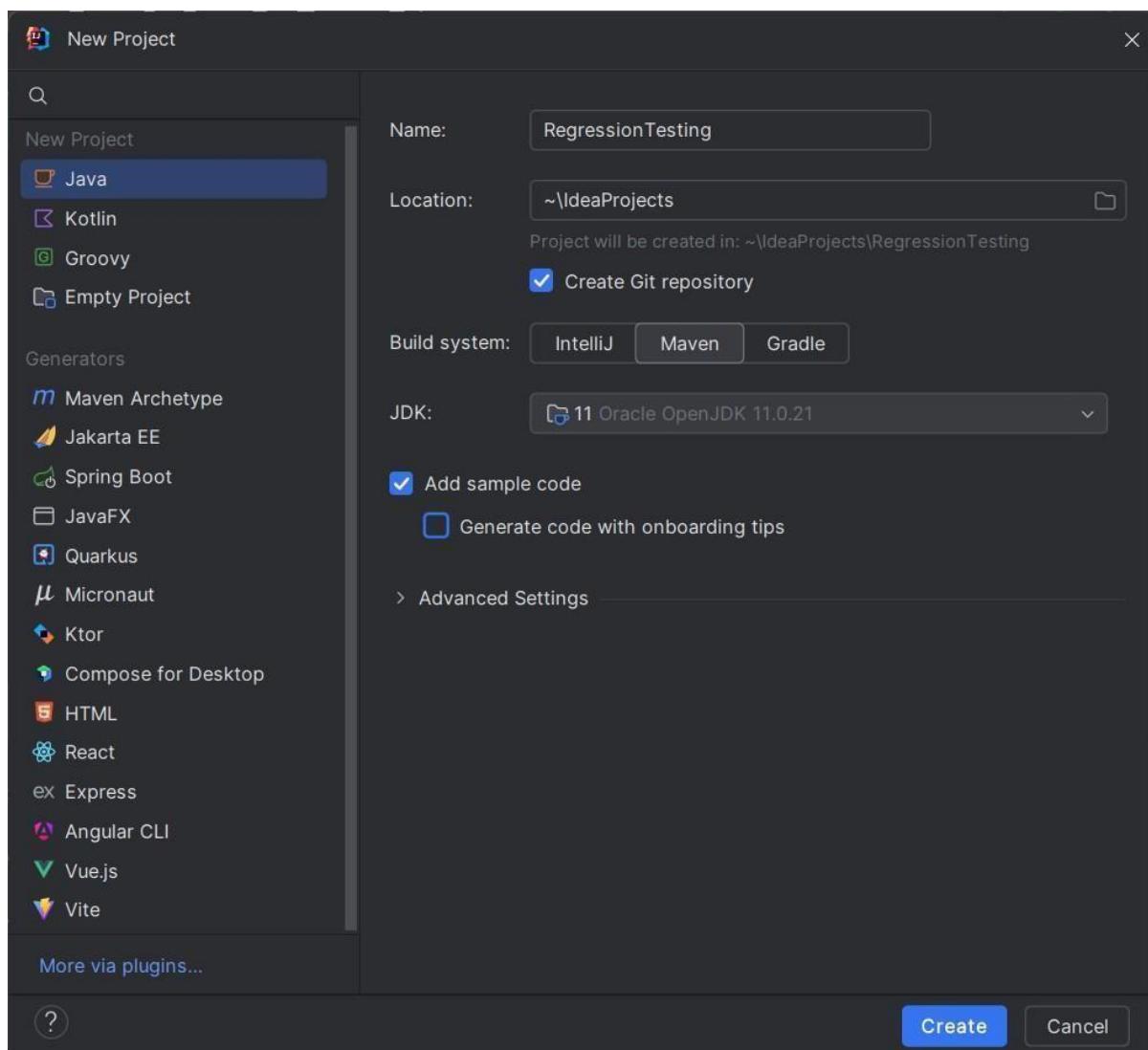
AIM

To run regression tests using the Maven build pipeline in Azure.

PROCEDURE

STEP 1: Create a new maven project in IntelliJ IDEA IDE.

1. Create a project with **RegressionTesting** as its name.
2. Choose the **Maven** build system and available JDK version.
3. Select the **Create Git Repository** option.



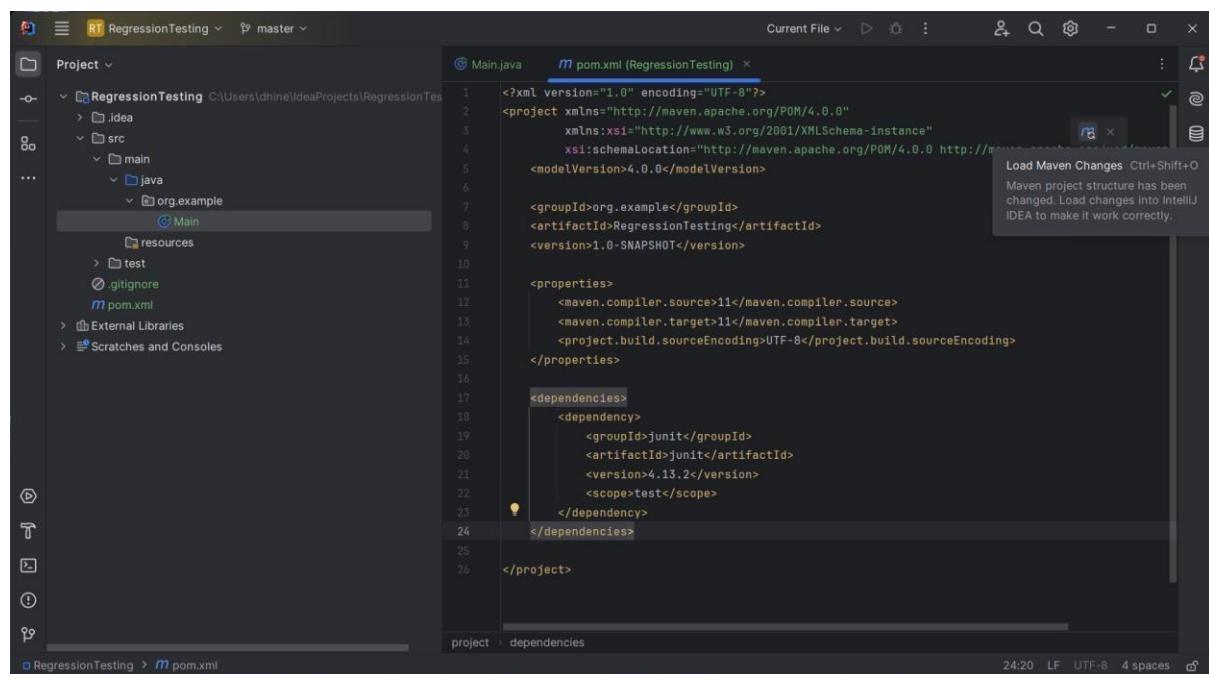
4. Wait for some time until the indexes of the project have been updated.
5. Add the following snippet of code into **pom.xml**

```

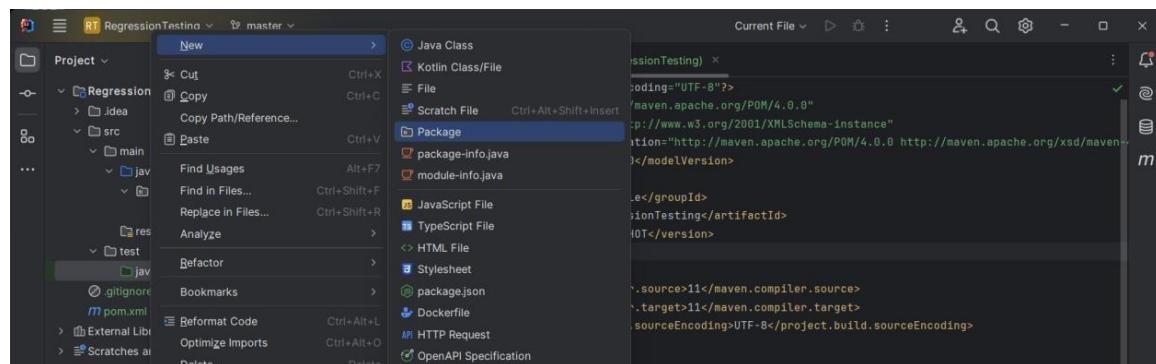
<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.13.2</version>
        <scope>test</scope>
    </dependency>
</dependencies>

```

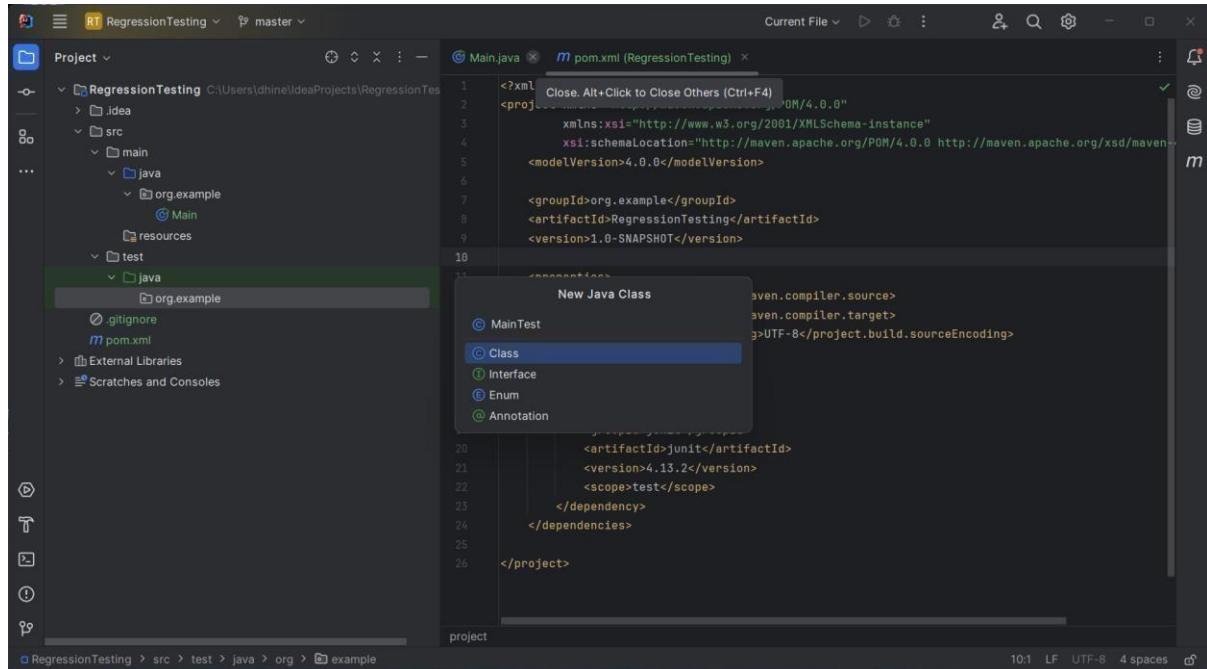
- Once the changes have been made click on the referred button to update the changes made to pom.xml or for simplicity just press **CTRL + SHIFT + O**



- Now, inside the java folder on the **Test** directory create a package named **org.example** (Right click on java folder inside the Test directory choose New and select package)



8. Right-click on the package and create a new class named **MainTest**



9. Paste the following into the MainTest.java file

```
package org.example;
import org.junit.Test;
import java.io.ByteArrayOutputStream;
import java.io.PrintStream;
import static org.junit.Assert.assertEquals;
public class MainTest {
    @Test
    public void testMainMethod() {
        // Redirect standard output to capture console output
        ByteArrayOutputStream outContent = new ByteArrayOutputStream();
        System.setOut(new PrintStream(outContent));

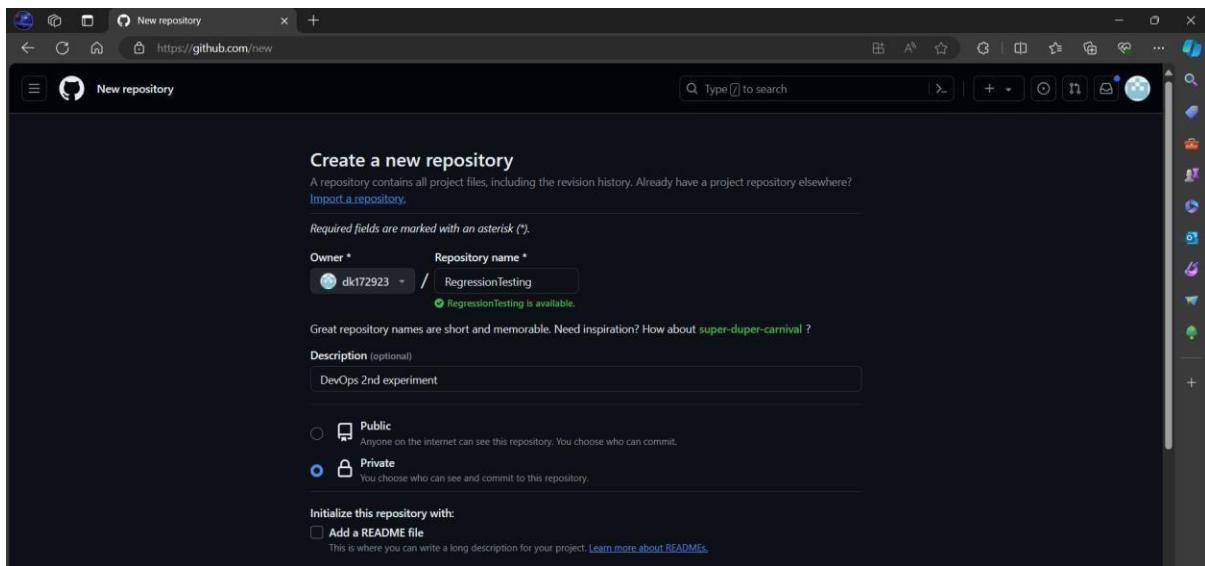
        // Call the main method
        Main.main(new String[]{});

        // Check if "Hello and welcome!" was printed
        assertEquals("Hello and welcome!", outContent.toString().trim());

        // Reset standard output
        System.setOut(System.out);
    }
}
```

STEP 2: Create a GitHub repository

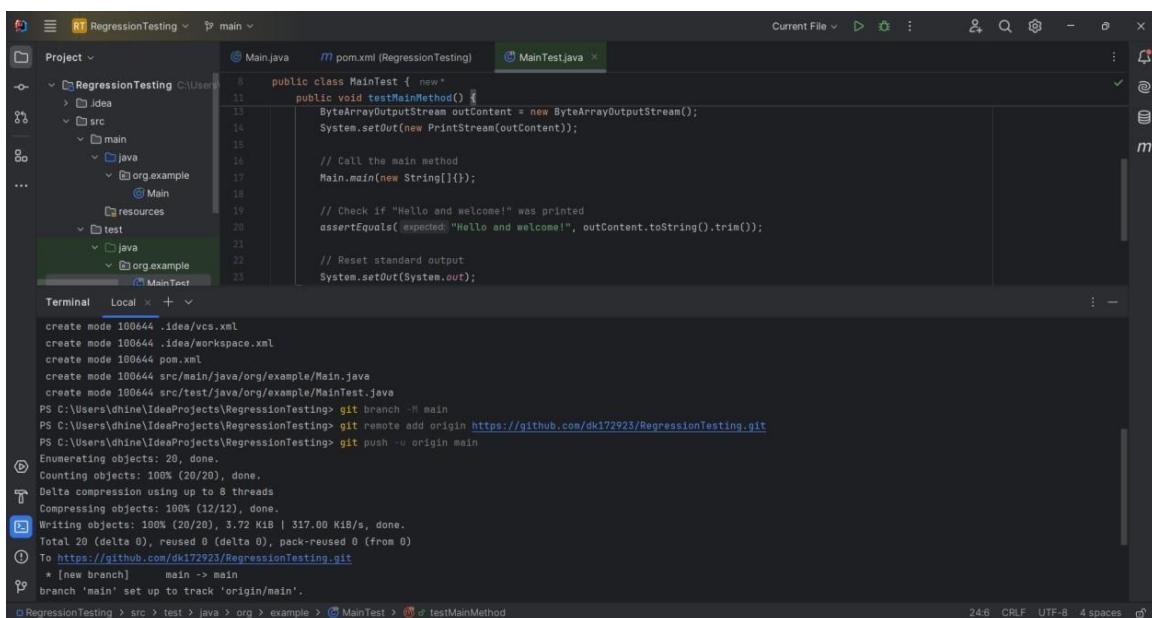
1. Create a private repository with RegressionTesting as its name.



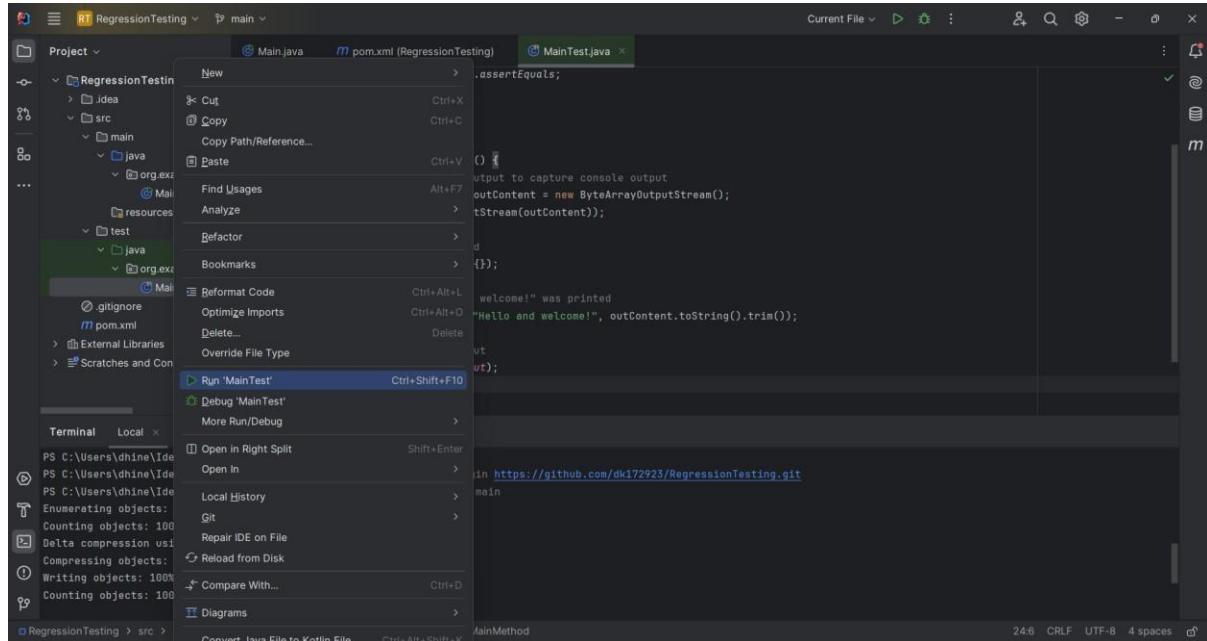
2. Execute the following commands in the terminal of IntelliJ IDEA

```
$ git init  
$ git add .  
$ git commit -m "regression"  
$ git branch -M main  
$ git remote add origin YOUR_REPOSITORY_LINK  
$ git push -u origin main
```

NOTE: YOUR_REPOSITORY_LINK must contain either the HTTPS or SSH link to your respective GitHub repository.



3. The MainTest class can be executed by Right-Clicking on the class from the project structure and choosing Run ‘MainTest’



The following can be achieved on the Run tab.

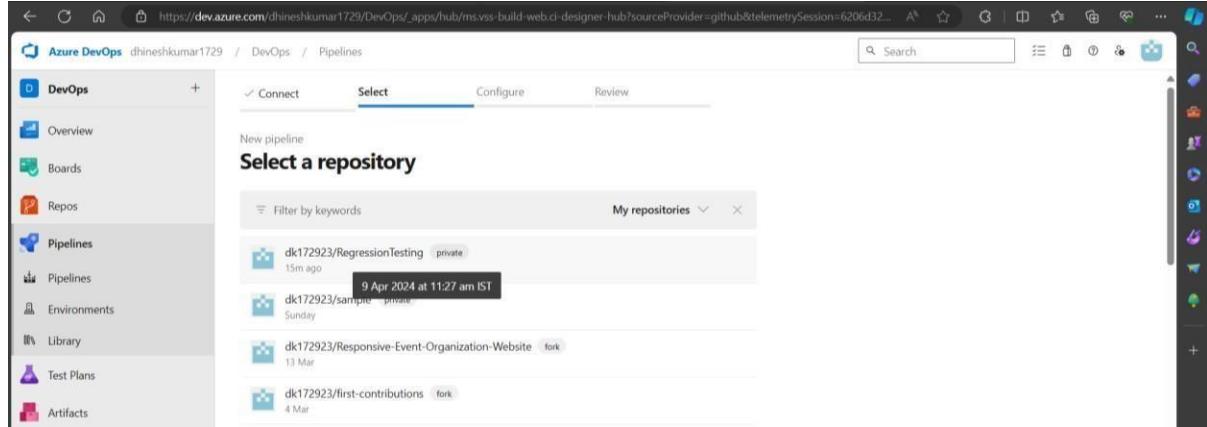


STEP 3: Create an Azure Pipeline

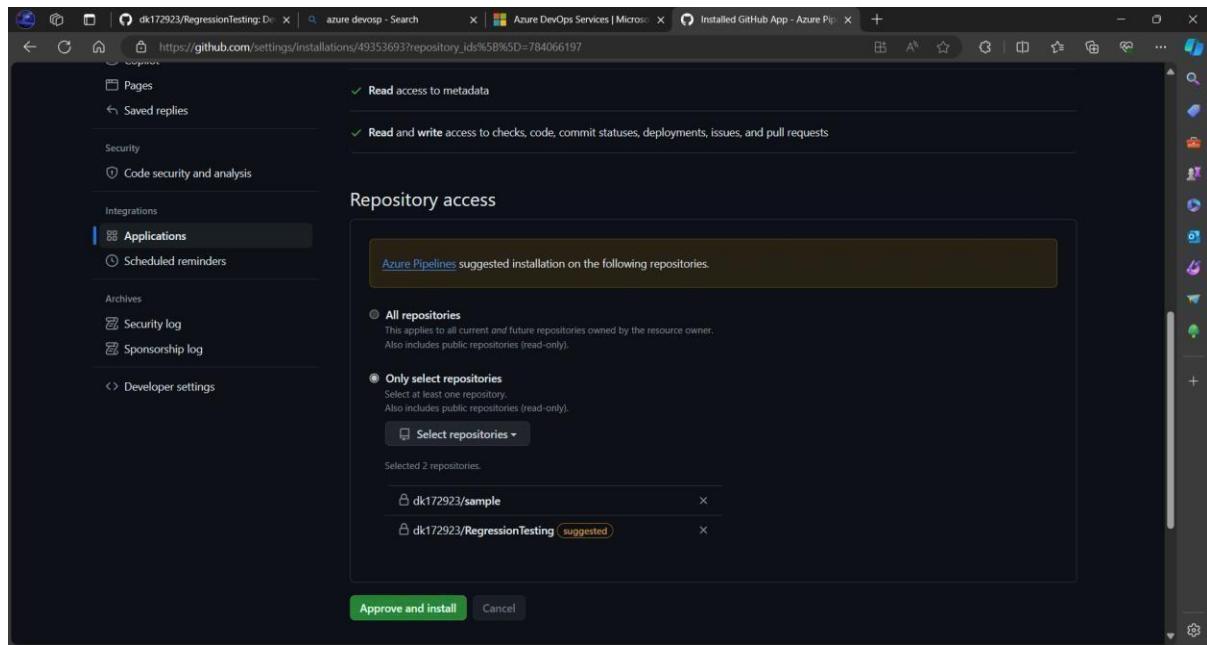
1. Open the Azure DevOps site

The screenshot shows the Azure DevOps site for the 'DevOps' project. The left sidebar has 'Pipelines' selected. The main area displays 'About this project' with the message 'A new project for the DevOps manual.' and 'Project stats' showing '100% Builds succeeded' over the last 7 days. The 'Members' section shows one user.

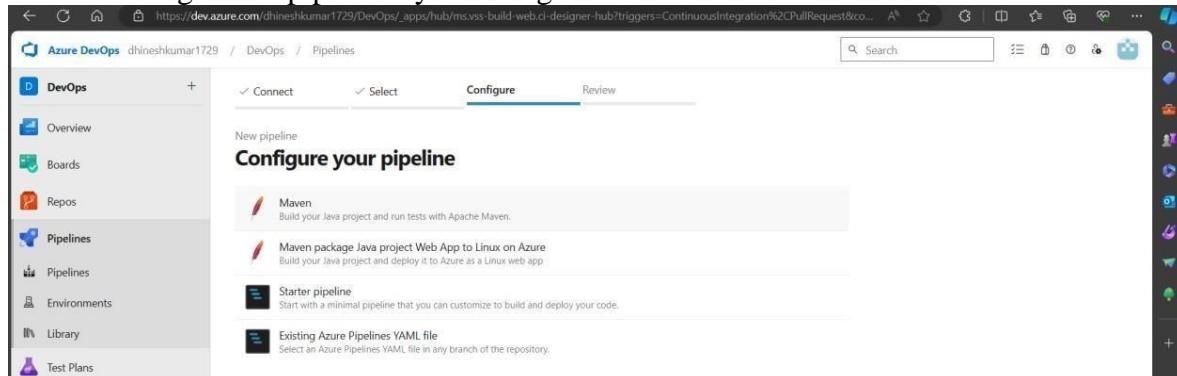
2. Click on existing project DevOps and choose Pipelines from the left pane.
3. Click on the **New Pipeline** button on the right.
4. Choose GitHub from the list.
5. Select the repository for RegressionTesting.



6. If prompted for permission then select Approve and Install on GitHub website.



7. Configure the pipeline by choosing Maven



8. On the next page an azure-pipeline.yml file is generated automatically. Replace this file with the following code

```
# Maven

# Build your Java project and run tests with Apache Maven.

# Add steps that analyze code, save build artifacts, deploy, and more:
# https://docs.microsoft.com/azure/devops/pipelines/languages/java

trigger:
- main

pool:
  vmImage: 'ubuntu-latest'

steps:
- task: Maven@3
  inputs:
    mavenPomFile: 'pom.xml'
    mavenOptions: '-Xmx3072m'
    javaHomeOption: 'JDKVersion'
    jdkVersionOption: '11'
    jdkArchitectureOption: 'x64'
    goals: 'clean test'

- task: PublishTestResults@2
  inputs:
    testResultsFiles: '**/surefire-reports/TEST-*.xml'
    testRunTitle: 'JUnit Test Results'
```

9. Click on the Save and Run button to commit the changes and execute the pipeline.

10. The following results can be obtained on the output page.

The screenshot shows the Azure DevOps interface for a pipeline job. The left sidebar is titled 'DevOps' and includes 'Overview', 'Boards', 'Repos', 'Pipelines', 'Environments', 'Library', 'Test Plans', and 'Artifacts'. The 'Pipelines' section is currently selected. The main content area is titled 'Jobs in run #20240409.1' and shows a single job named 'dk17293.RegressionTesting'. The job details are as follows:

- Pool: Azure Pipelines
- Image: ubuntu-latest
- Agent: Hosted Agent
- Started: Today at 11:51 am
- Duration: 22s
- Job preparation parameters
- 100% tests passed

The job steps listed are: Initialize job (2s), Checkout dk17293/Re... (1s), Maven (15s), PublishTestResults (3s), Post-job: Checkout dk... (<1s), and Finalize Job (<1s). All steps are marked with green checkmarks.

11. From the test results tab it can be found that the tests passed.

The screenshot shows the Azure DevOps interface for the 'Test results' tab of a pipeline run. The left sidebar is identical to the previous screenshot. The main content area is titled '#20240409.1 • Set up CI with Azure Pipelines' and shows the following summary:

This run is being retained as one of 3 recent runs by pipeline. [View retention leases](#)

Summary [Tests](#) [Code Coverage](#)

2 Run(s) Completed (2 Passed, 0 Failed)

Total tests	Passed	Failed	Others	Pass percentage	Run duration	Tests not reported
2	2	0	0	100%	16ms	0

2 Passed, 0 Failed, 0 Others, 100% Pass percentage, 16ms Run duration, 0 Tests not reported.

Filter by test or run name [Test run](#) [Column Options](#)

RESULT

Thus, regression tests have been run using the Maven Build Pipeline in Azure.

Ex. No: 3

INSTALL JENKINS IN CLOUD

Date:

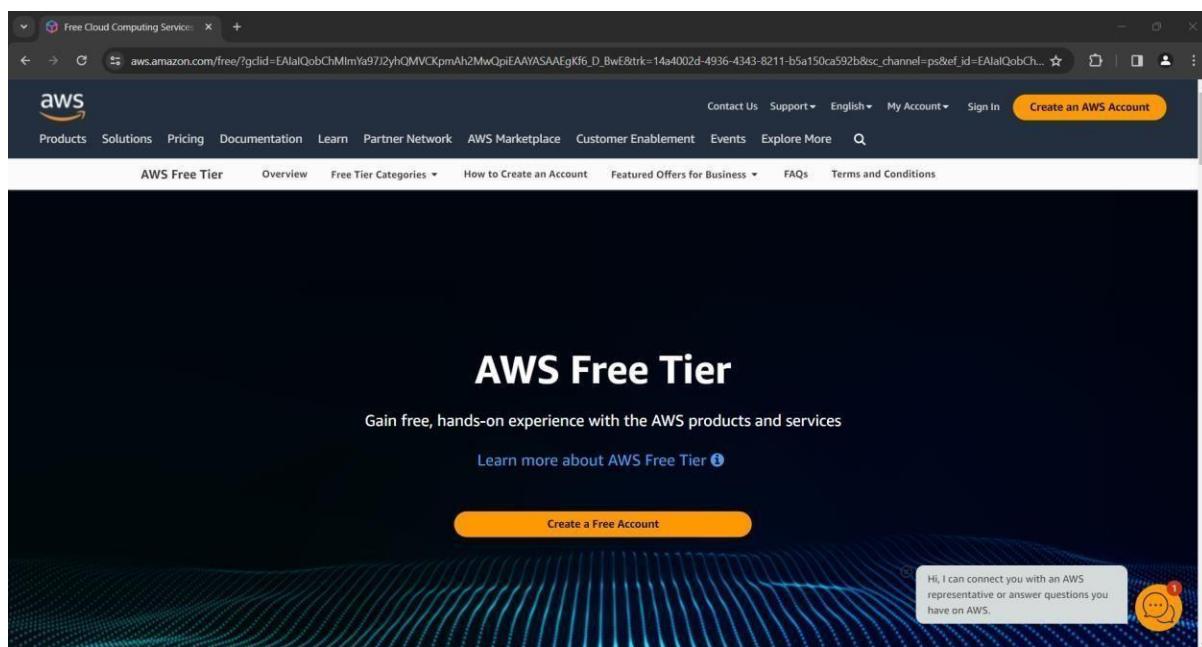
AIM

To install Jenkins in the cloud (AWS).

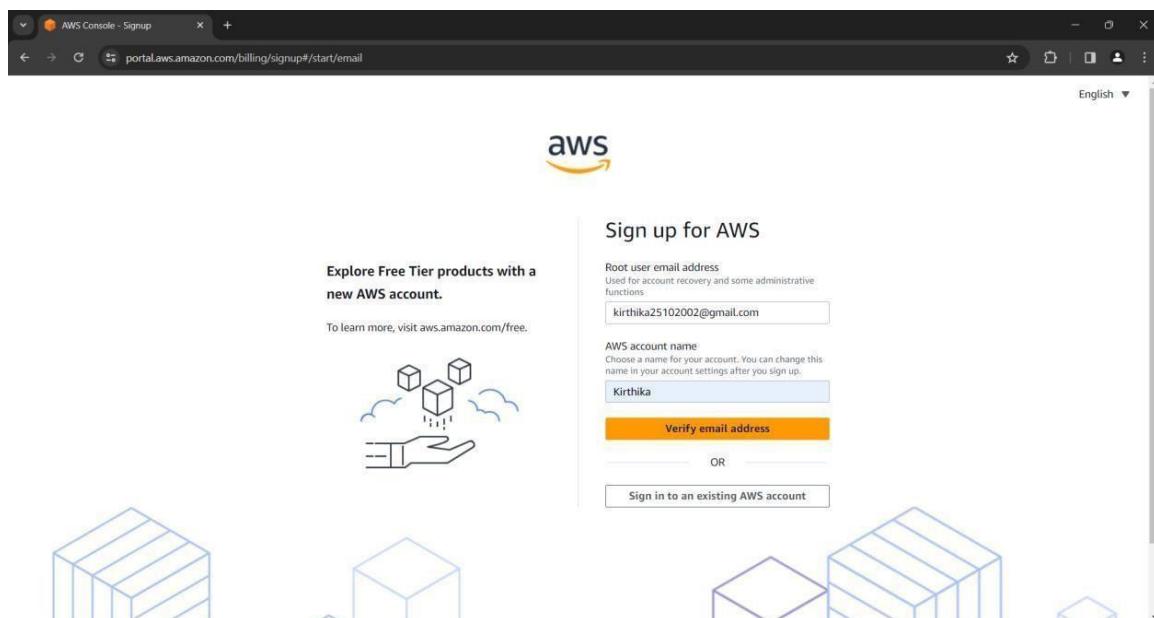
PROCEDURE

STEP 1: Create an AWS account

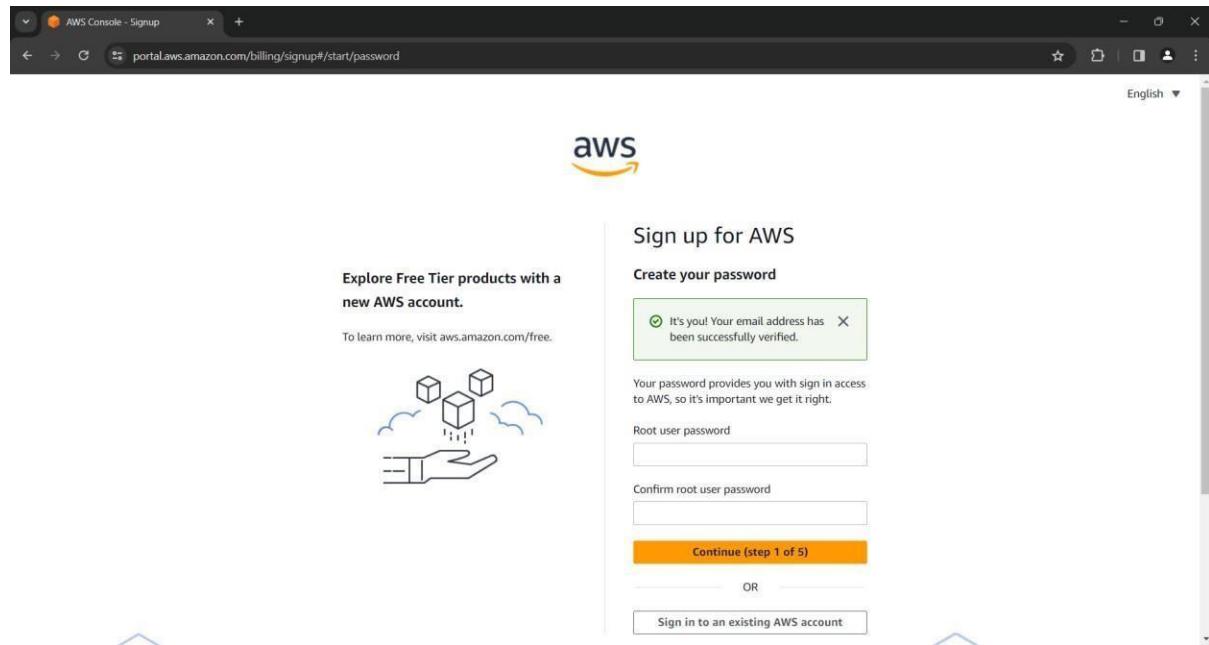
1. Go to <https://aws.amazon.com/> and create an AWS Account.



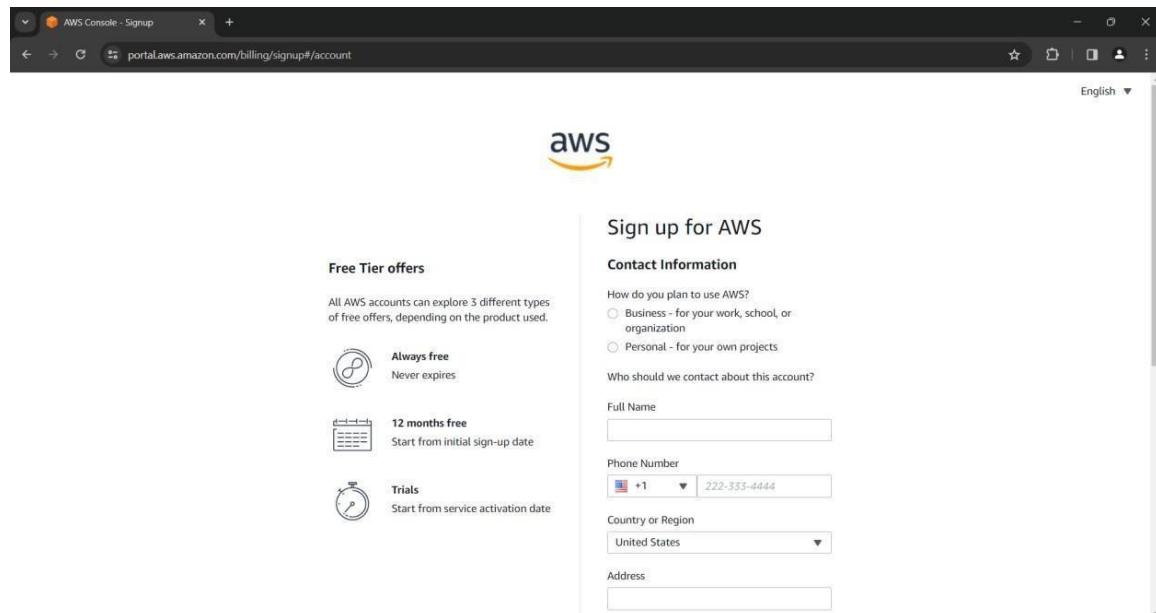
2. Provide a root email ID and username for the user.



3. After Email verification set up your password.



4. Complete the sign-up process by providing additional information.



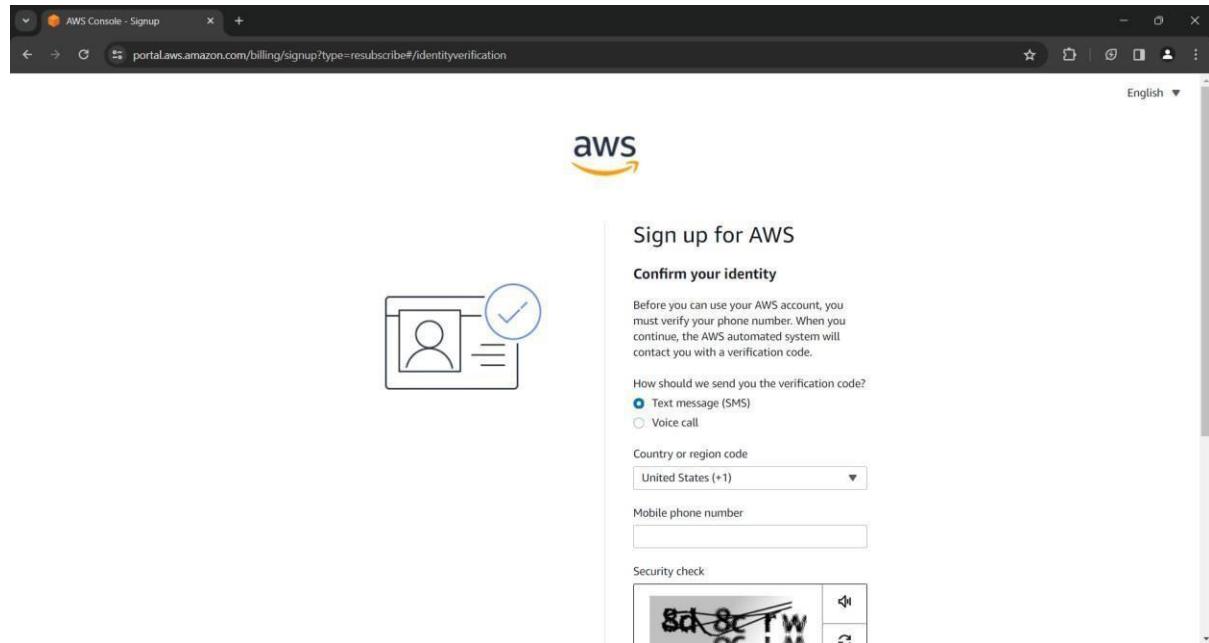
5. Provide the debit card information for the identity verification process. A charge of 2 rupees will be debited.

The screenshot shows the 'Sign up for AWS' page on the AWS Console. The 'Secure verification' section contains a note: 'We will not charge you for usage below AWS Free Tier limits. We may temporarily hold up to \$1 USD (or an equivalent amount in local currency) as a pending transaction for 3-5 days to verify your identity.' Below this is a shield icon with a checkmark. The 'Billing Information' section includes fields for 'Credit or Debit card number' (with a required field note), logos for VISA, MasterCard, American Express, and RuPay, and a note that AWS accepts most major credit and debit cards. It also includes fields for 'Expiration date' (Month and Year dropdowns), 'Security code (CVV/CVC)', 'Cardholder's name', and a checkbox for saving card information.

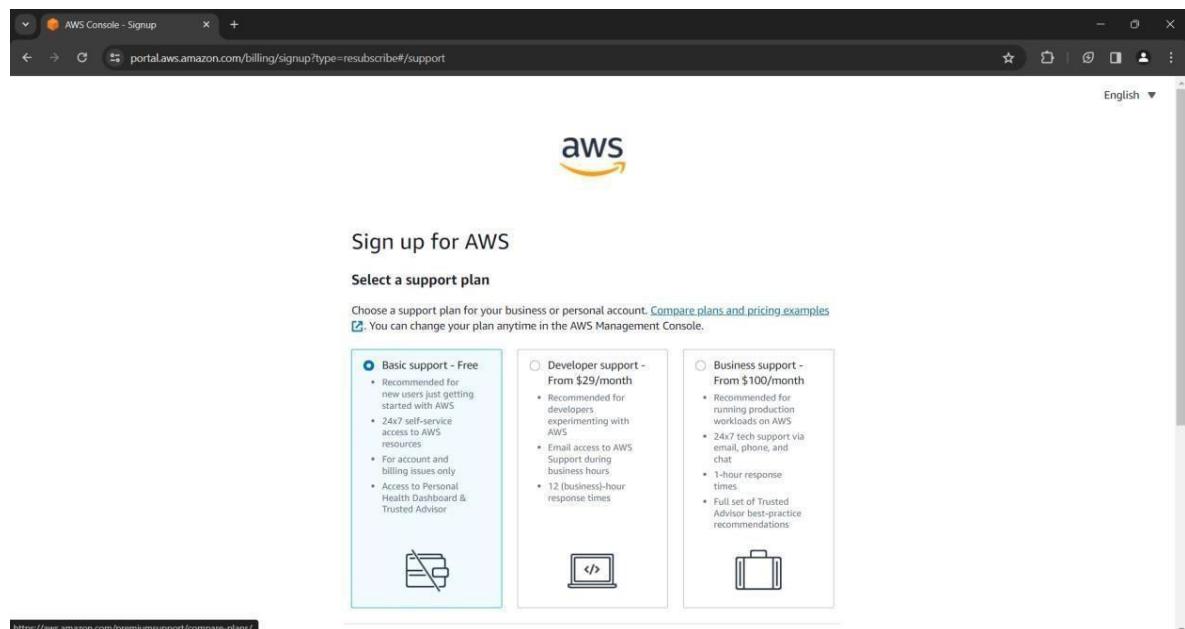
6. Confirm your identity by providing the PAN card number associated with the name of the user.

The screenshot shows the 'Sign up for AWS' page on the AWS Console, specifically the 'Confirm your identity' section. It features a profile icon with a checkmark. The 'Name' field is populated with 'Kirthika B'. The 'Primary purpose of account registration' dropdown is set to 'Personal use'. The 'Ownership type' dropdown is set to 'Individual'. The 'India document type' dropdown is set to 'Permanent Account Number (PAN)'. There is a note: 'To verify your identity, the name on the document must match the name that you chose.'

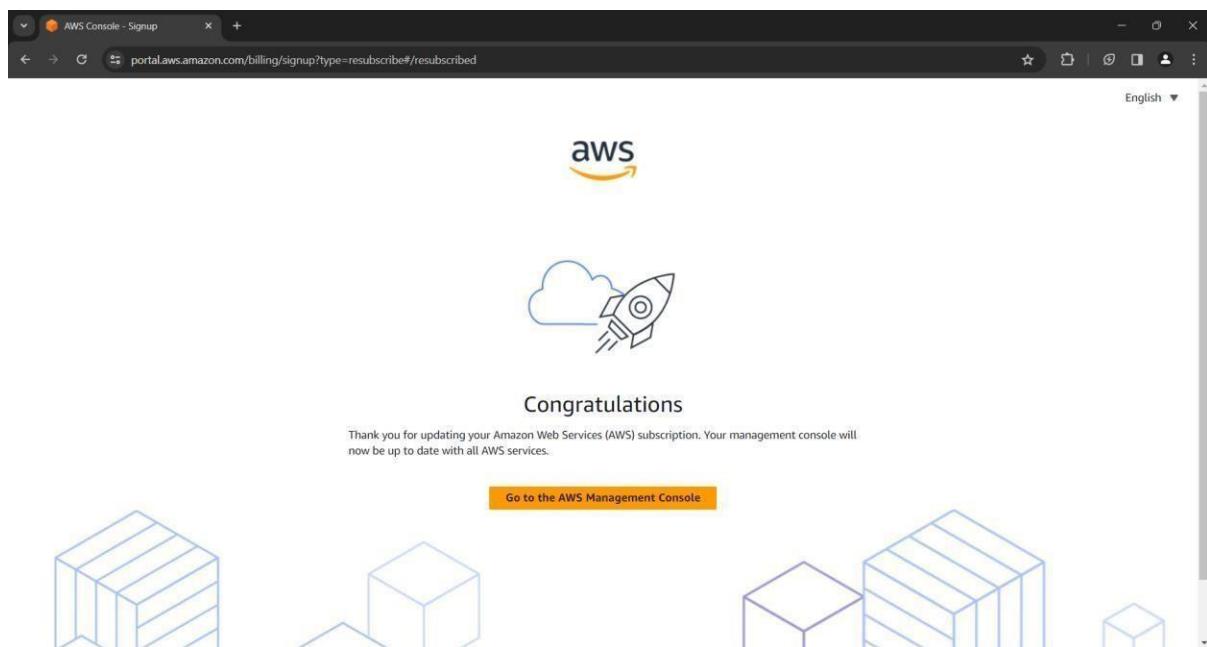
7. Verify the phone number and perform a security check.



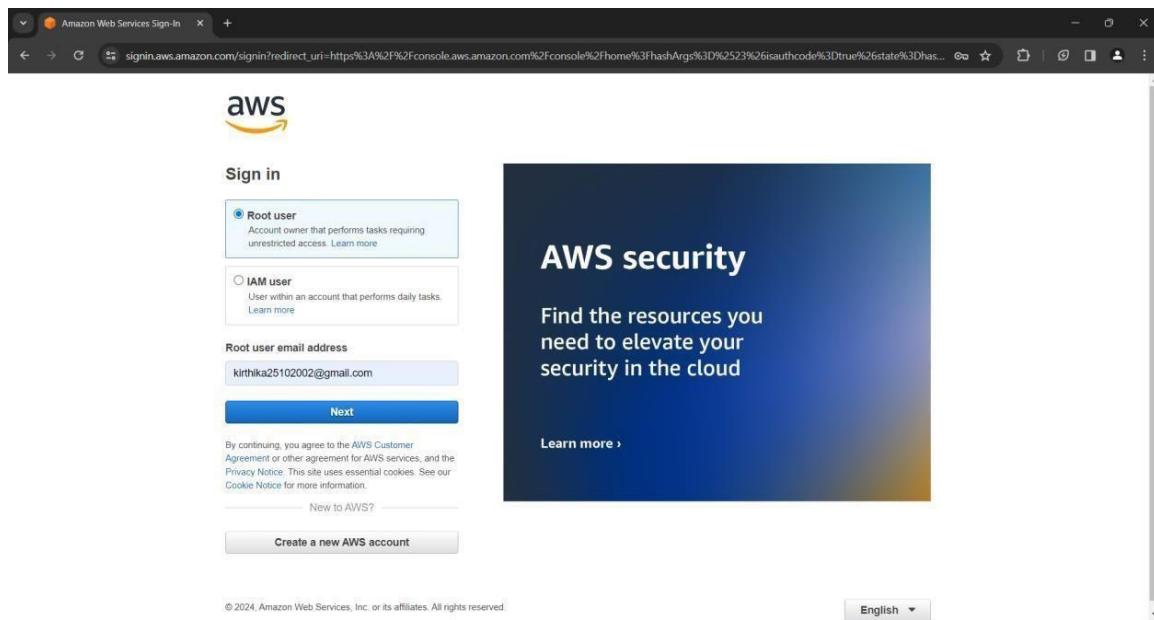
8. Select Basic Support as the plan since it is free of charge.



9. The AWS account has been successfully created and verified. Now click on **Go to the Aws Management Console** button.

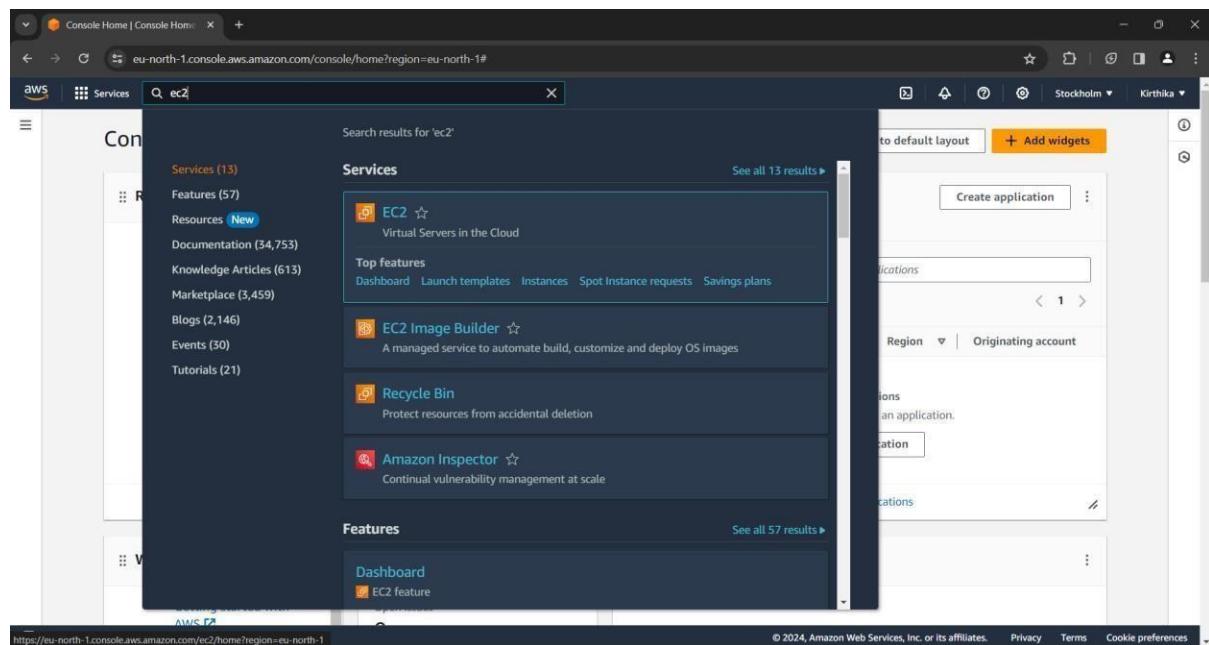


10. Sign in as a Root user by providing the appropriate email address and password.

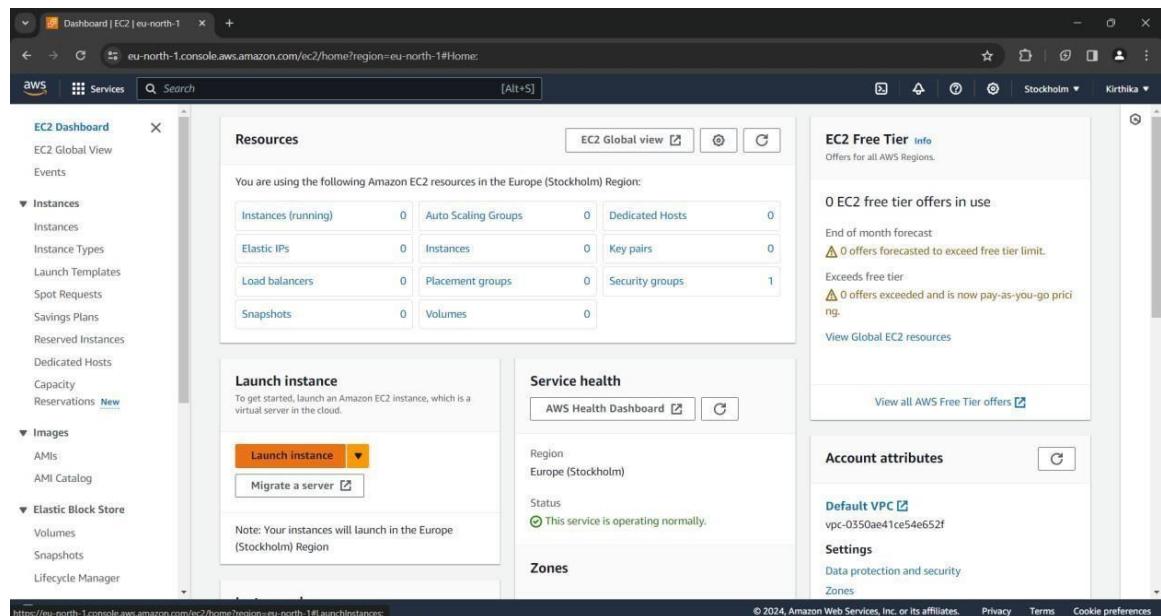


STEP 2: Create an EC2 instance.

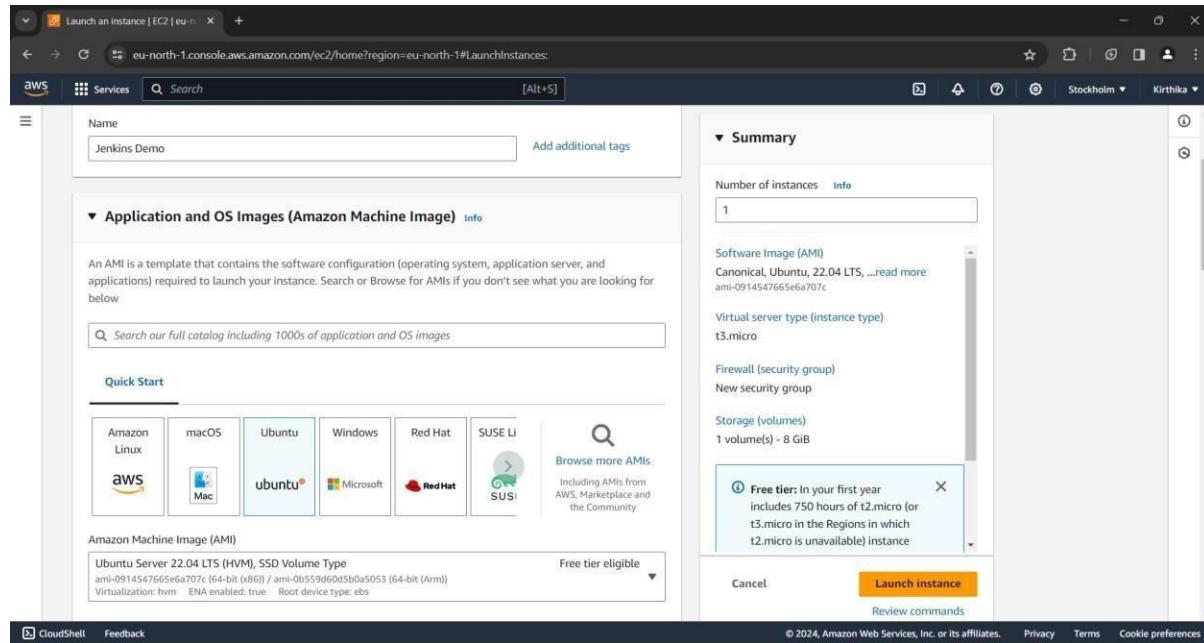
1. Search EC2 in the search bar and choose EC2 from the services tab.



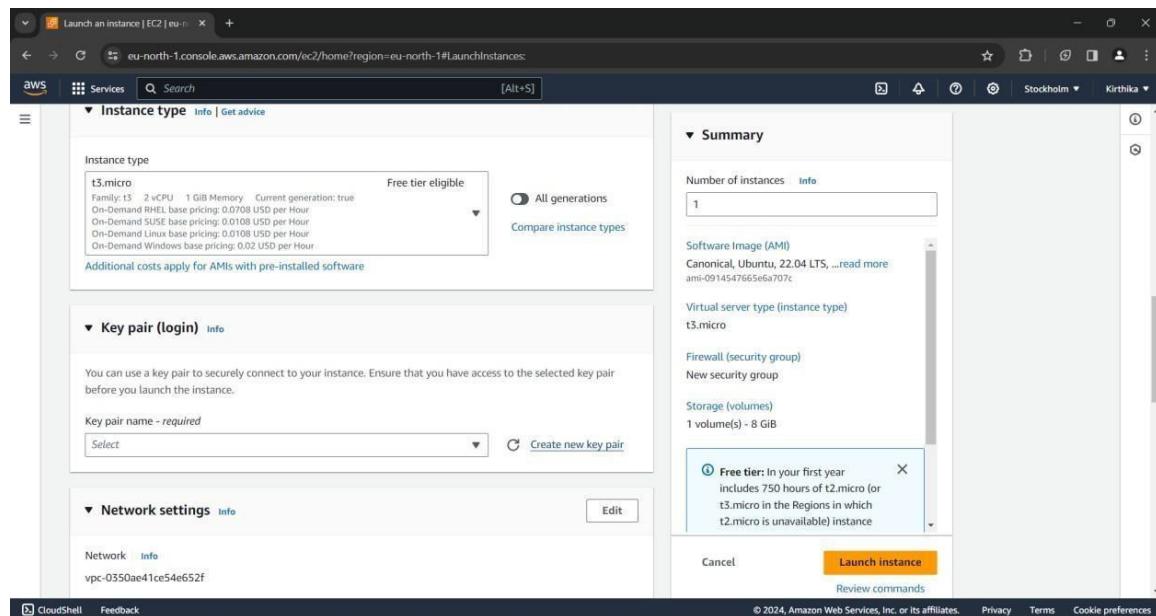
2. Click on the **Launch Instance** button.



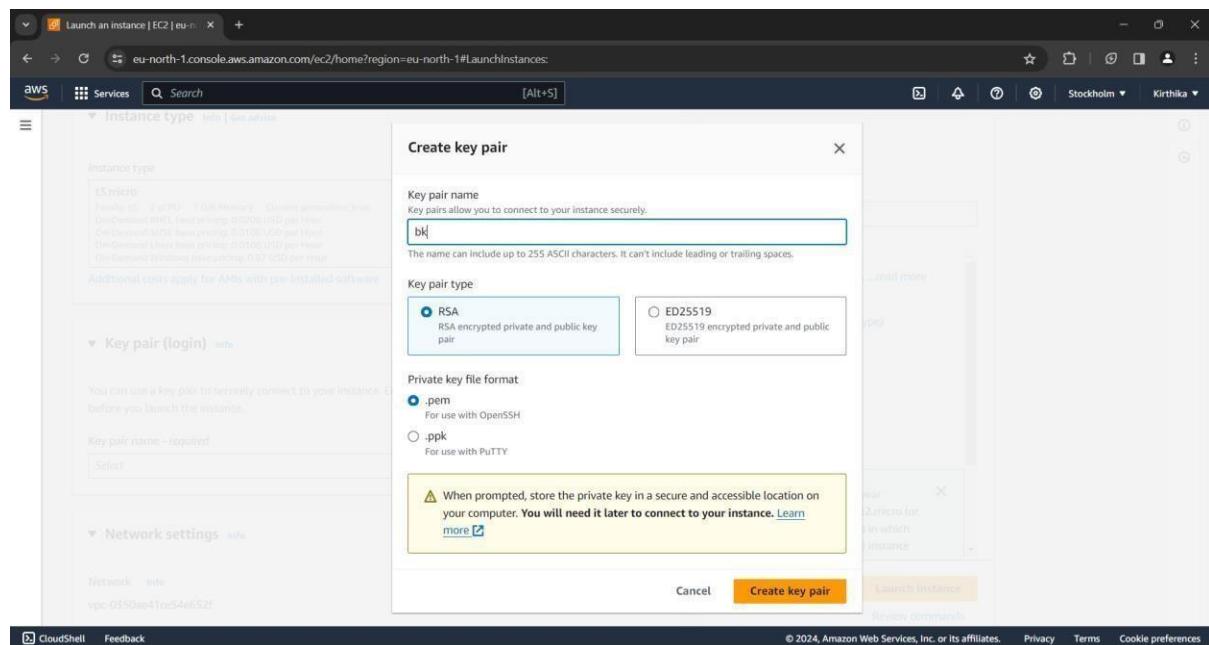
- Provide a name for the instance for example **Jenkins Demo** and choose Ubuntu as the operating system.



- Create a new key pair if you haven't created one yet.

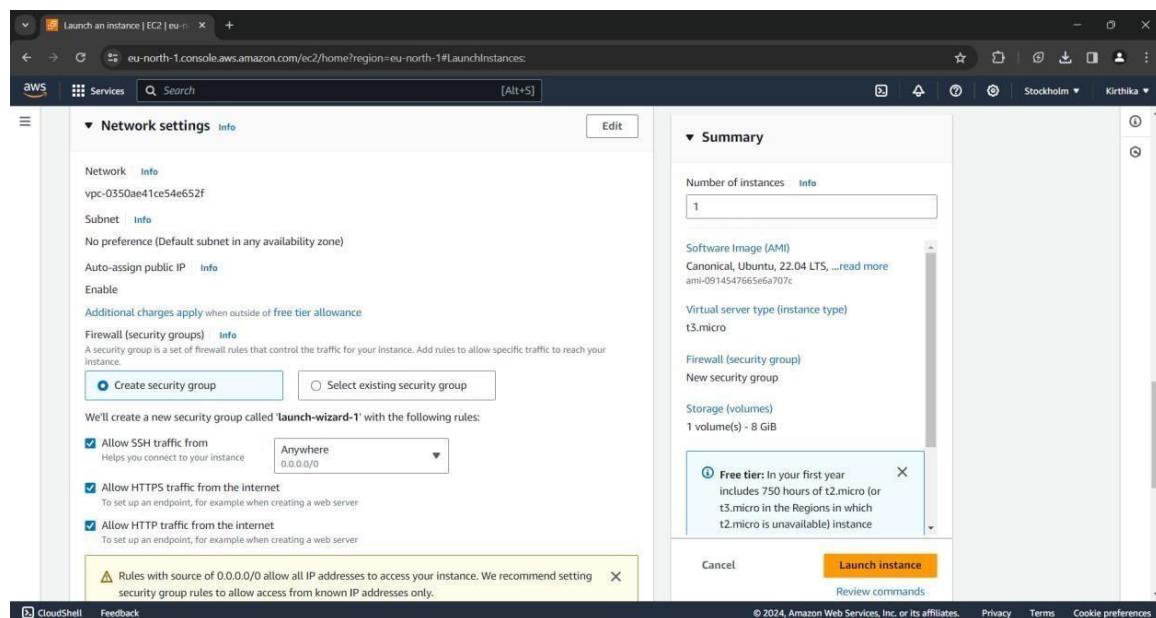


5. Give a suitable name for the key pair, select the key pair type as RSA and click on the **Create key pair** button at the bottom.

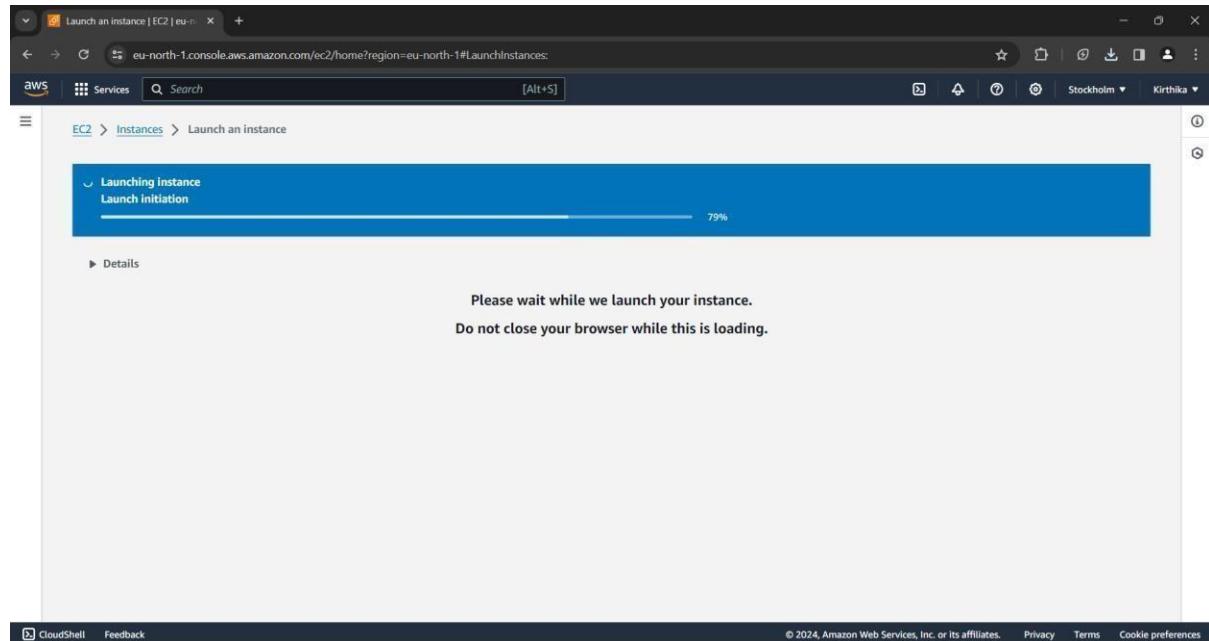


6. In the Network Settings tab select the following options.

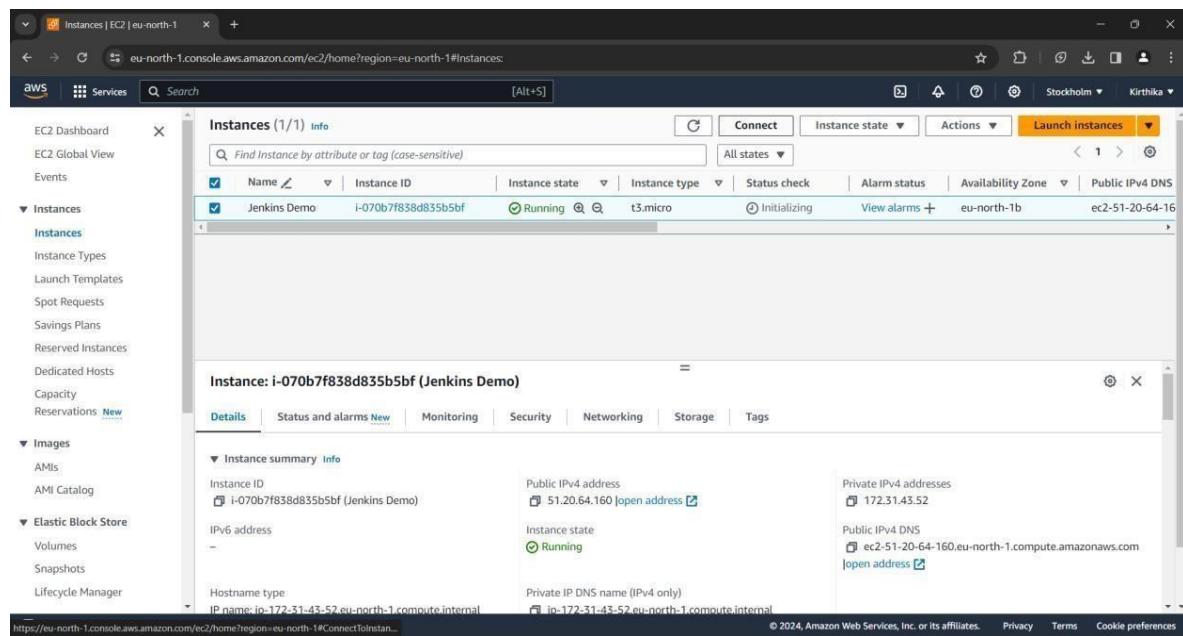
- Allow HTTPS traffic from the internet
- Allow HTTP traffic from the internet



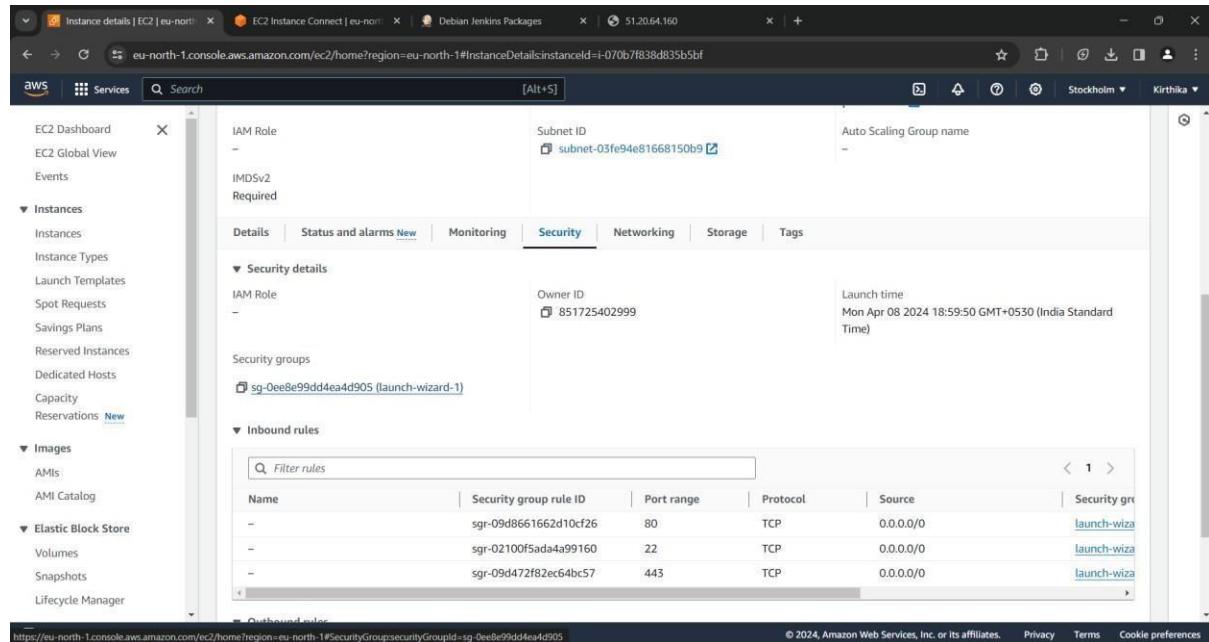
7. Wait while the instance is being launched. It may take some time.



8. Select the Jenkins Demo instance and click on the **Connect** button.



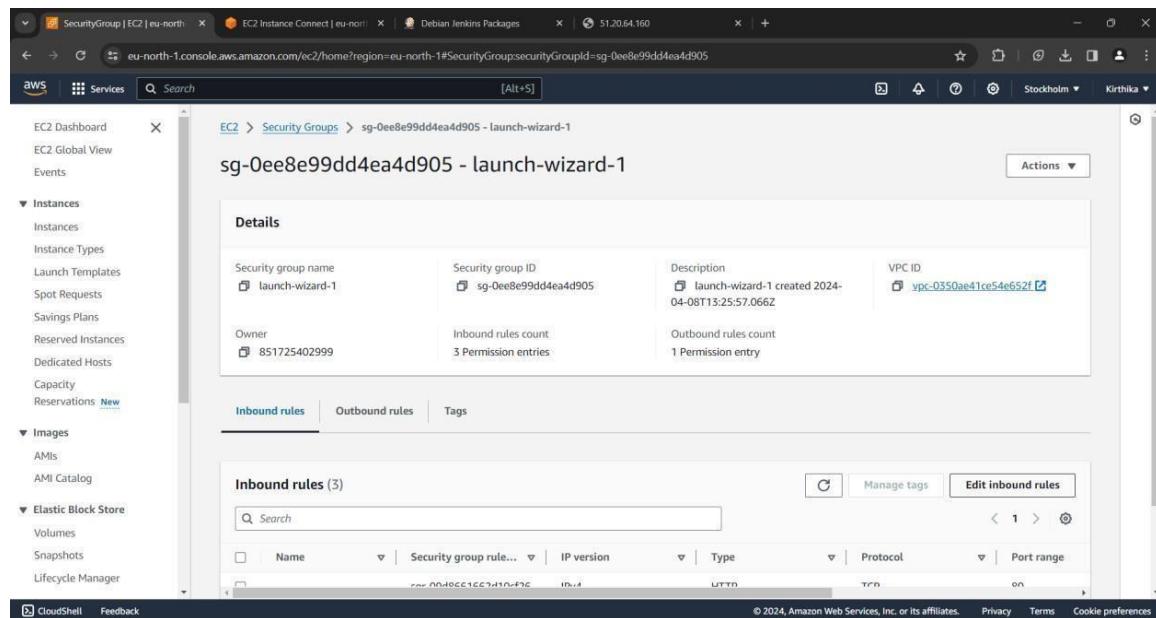
9. Click on the Security tab and choose the **Security Groups** option.



The screenshot shows the AWS EC2 Instances details page. The left sidebar has sections for EC2 Dashboard, EC2 Global View, Events, Instances (with sub-options like Instances Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity, and Reservations), Images (AMIs, AMI Catalog), and Elastic Block Store (Volumes, Snapshots, Lifecycle Manager). The main content area has tabs for IAM Role, Subnet ID (subnet-03fe94e81668150b9), Auto Scaling Group name, and Security (which is currently selected). Under Security, there are sections for Security details (IAM Role, Owner ID 851725402999, Launch time Mon Apr 08 2024 18:59:50 GMT+0530 (India Standard Time)), Security groups (sg-0ee8e99dd4ea4d905 (launch-wizard-1)), and Inbound rules. The Inbound rules table shows three entries:

Name	Security group rule ID	Port range	Protocol	Source	Security group
-	sgr-09d8661662d10cf26	80	TCP	0.0.0.0/0	launch-wiza
-	sgr-02100f5ada4a99160	22	TCP	0.0.0.0/0	launch-wiza
-	sgr-09d472ff82ec64bc57	443	TCP	0.0.0.0/0	launch-i-wiza

10. Click on the **Edit Inbound Rules** button.



The screenshot shows the AWS Security Groups details page for sg-0ee8e99dd4ea4d905 - launch-wizard-1. The left sidebar is identical to the previous screenshot. The main content area shows the Details section with fields for Security group name (launch-wizard-1), Security group ID (sg-0ee8e99dd4ea4d905), Description (launch-wizard-1 created 2024-04-08T15:25:57.066Z), Owner (851725402999), Inbound rules count (3 Permission entries), and Outbound rules count (1 Permission entry). Below this is the Inbound rules section, which lists the same three entries as the previous screenshot:

Name	Security group rule ID	Port range	Protocol	Source	Security group
-	sgr-09d8661662d10cf26	80	TCP	0.0.0.0/0	launch-wiza
-	sgr-02100f5ada4a99160	22	TCP	0.0.0.0/0	launch-wiza
-	sgr-09d472ff82ec64bc57	443	TCP	0.0.0.0/0	launch-i-wiza

11. Click on **Add rule** at the left bottom, enter the port range as 8080 and choose 0.0.0.0/0

The screenshot shows the AWS Management Console with the URL eu-north-1.console.aws.amazon.com/ec2/home?region=eu-north-1#ModifyInboundSecurityGroupRules:securityGroupId=sg-0ee8e99dd4ea4d905. The page displays a table of security group rules. A new rule is being added at the bottom:

Security group rule ID	Type	Protocol	Port range	Source	Description - optional
sgr-09d8661662d10cf26	HTTP	TCP	80	Custom	<input type="text" value="0.0.0.0"/> <input type="button" value="X"/>
sgr-02100f5ada4a99160	SSH	TCP	22	Custom	<input type="text" value="0.0.0.0"/> <input type="button" value="X"/>
sgr-09d472f82ec64bc57	HTTPS	TCP	443	Custom	<input type="text" value="0.0.0.0"/> <input type="button" value="X"/>
-	Custom TCP	TCP	8080	Anyw...	<input type="text" value="0.0.0.0"/> <input type="button" value="X"/>

A yellow warning message at the bottom states: "⚠ Rules with source of 0.0.0.0/0 or ::/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only." Below the table are buttons for "Add rule", "Cancel", "Preview changes", and "Save rules".

12. Click on the Connect button at the bottom after choosing Connect using EC2 Instance Connect.

The screenshot shows the AWS Management Console with the URL eu-north-1.console.aws.amazon.com/ec2/home?region=eu-north-1#ConnectToInstance:instanceId=i-070b7f838d835b5bf. The page displays a "Connect to instance" form for an instance with ID **i-070b7f838d835b5bf (Jenkins Demo)**.

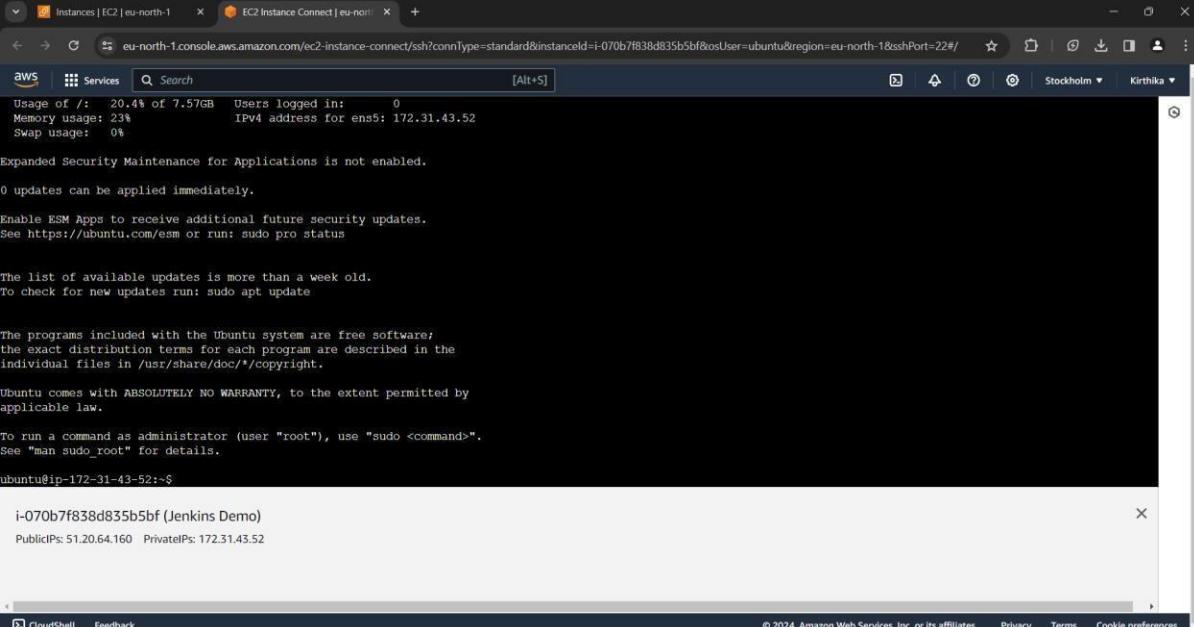
The "EC2 Instance Connect" tab is selected. The "Connection Type" section shows two options:

- Connect using EC2 Instance Connect: "Connect using the EC2 Instance Connect browser-based client, with a public IPv4 address."
- Connect using EC2 Instance Connect Endpoint: "Connect using the EC2 Instance Connect browser-based client, with a private IPv4 address and a VPC endpoint."

Below the connection type, the "Public IP address" is listed as **51.20.64.160**. The "Username" field contains **ubuntu**. A note at the bottom states: "Note: In most cases, the default username, ubuntu, is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username."

At the bottom of the page are buttons for "CloudShell", "Feedback", "Cancel", "Preview changes", and "Save changes".

13. A terminal is opened on a new tab in the same browser.



```
Usage of /: 20.4% of 7.57GB Users logged in: 0
Memory usage: 23% IPv4 address for ens5: 172.31.43.52
Swap usage: 0%
Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

The list of available updates is more than a week old.
To check for new updates run: sudo apt update

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

to run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

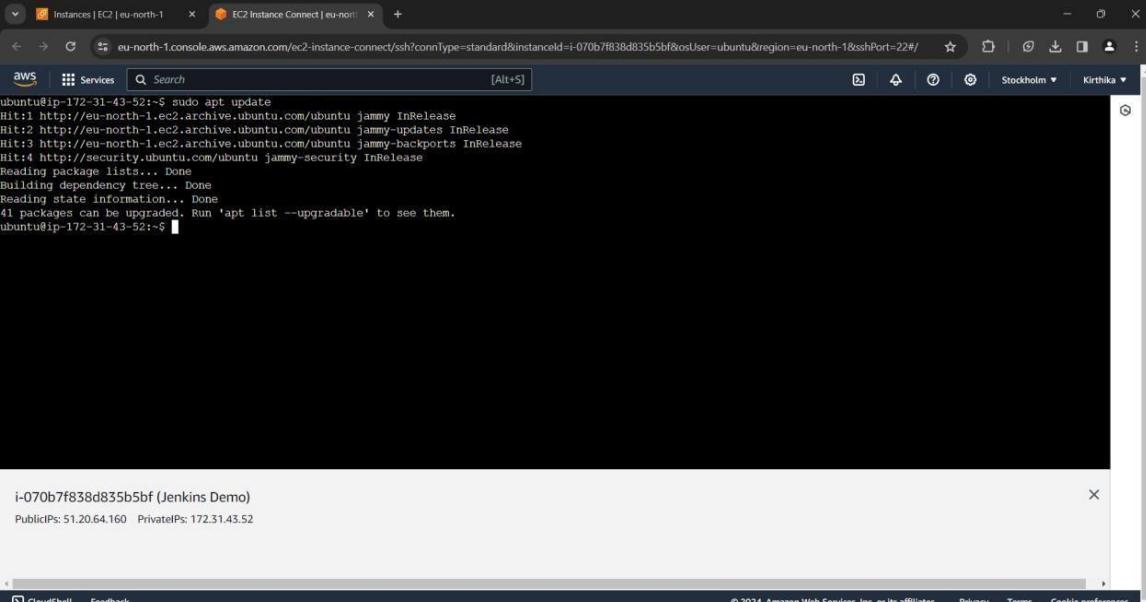
ubuntu@ip-172-31-43-52:~
```

i-070b7f838d835b5bf (Jenkins Demo)
PublicIPs: 51.20.64.160 PrivateIPs: 172.31.43.52

STEP 3: Installing Jenkins on the instance.

1. Execute the following commands on the terminal to update the OS.

```
$ clear  
$ sudo apt update
```



```
ubuntu@ip-172-31-43-52:~$ sudo apt update
Hit:1 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu jammy InRelease
Hit:2 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:3 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu jammy-backports InRelease
Hit:4 http://security.ubuntu.com/ubuntu jammy-security InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
41 packages can be upgraded. Run 'apt list --upgradable' to see them.
ubuntu@ip-172-31-43-52:~$
```

i-070b7f838d835b5bf (Jenkins Demo)
PublicIPs: 51.20.64.160 PrivateIPs: 172.31.43.52

2. Execute the following command to install Jenkins.

```
$ sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
```

```
$ echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
/etc/apt/sources.list.d/jenkins.list >/dev/null
```

```
$ sudo apt-get update
```

```
$ sudo apt-get install fontconfig openjdk-17-jre
```

```
$ sudo apt-get install Jenkins
```

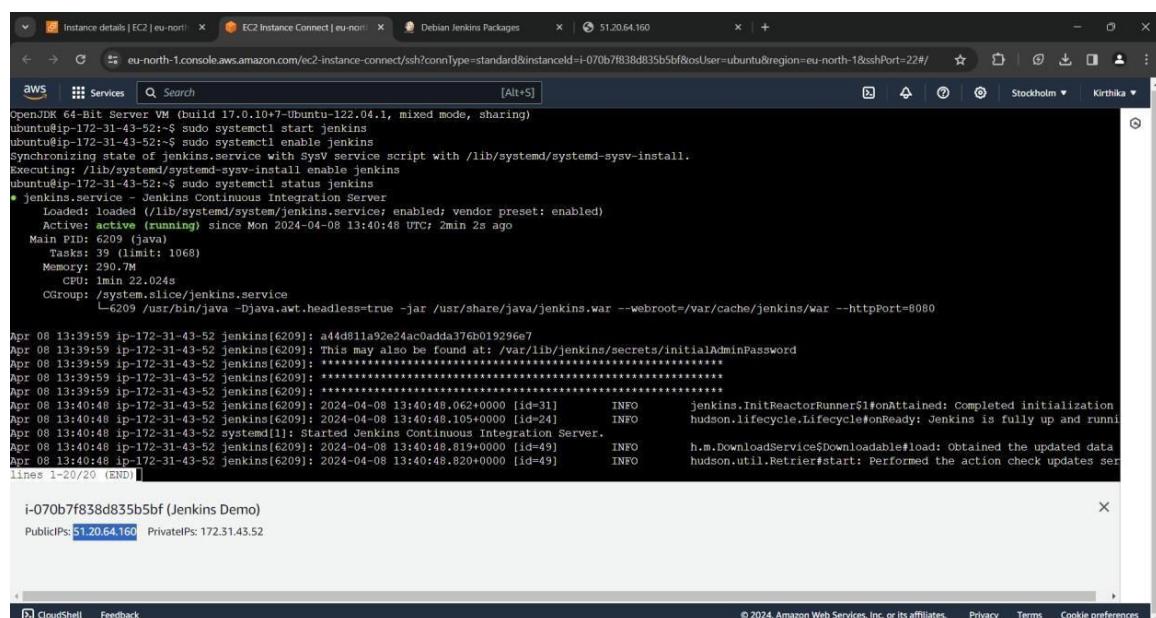
These commands can also be found at <https://pkg.jenkins.io/debian-stable/> for Debian releases.

3. The following commands will start Jenkins.

```
$ sudo systemctl start Jenkins
$ sudo systemctl enable Jenkins
$ sudo systemctl status Jenkins
```

4. Copy the public IP address found below the terminal and copy the address. Open a new tab, paste the IP address and add :**8080** at the last.

Example: 51.20.64.160:8080



A screenshot of a terminal window titled "Debian Jenkins Packages". The window shows the output of several commands:

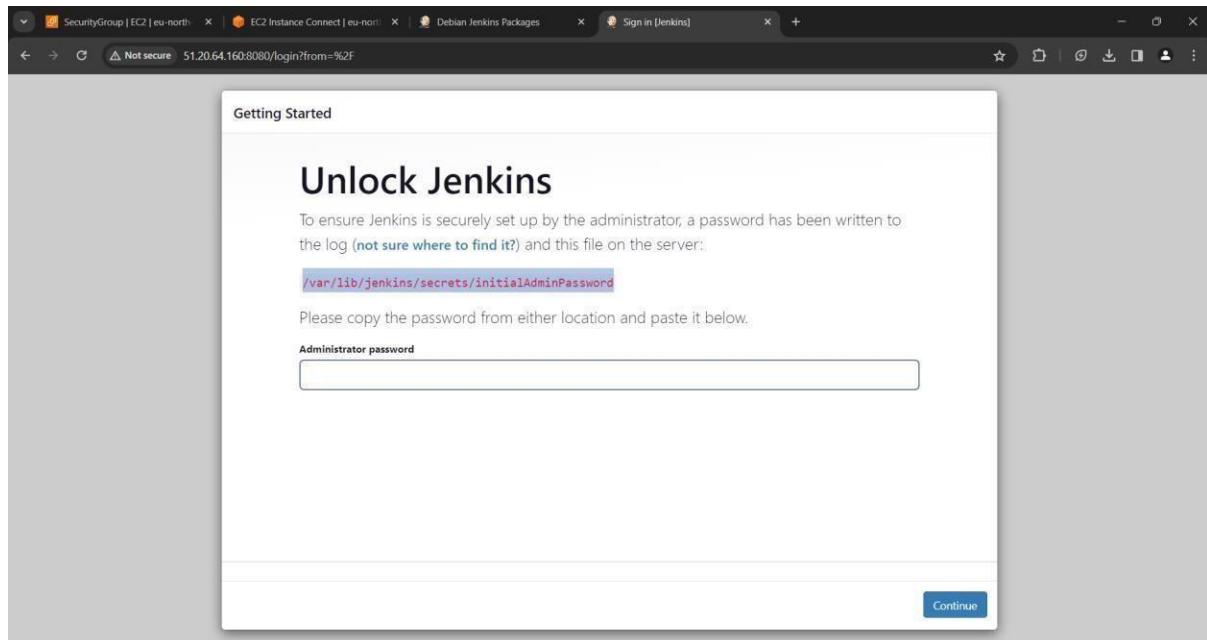
```
OpenJDK 64-Bit Server VM (builid 17.0.10+7-Ubuntu-122.04.1, mixed mode, sharing)
ubuntu@ip-172-31-43-52:~$ sudo systemctl start jenkins
ubuntu@ip-172-31-43-52:~$ sudo systemctl enable jenkins
Synchronizing state of jenkins.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable jenkins
ubuntu@ip-172-31-43-52:~$ sudo systemctl status jenkins
● jenkins.service - Jenkins continuous Integration Server
  Loaded: loaded (/lib/systemd/system/jenkins.service; enabled; vendor preset: enabled)
  Active: active (running) since Mon 2024-04-08 13:40:48 UTC; 2min 2s ago
    Main PID: 6209 (java)
       Tasks: 39 (limit: 1068)
      Memory: 290.7M
         CPU: 1m1n 22.024s
        CGroup: /system.slice/jenkins.service
                └─ 6209 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war --httpPort=8080

Apr 08 13:39:59 ip-172-31-43-52 jenkins[6209]: a44d811a92e24ac0adda376b019296e7
Apr 08 13:39:59 ip-172-31-43-52 jenkins[6209]: This may also be found at: /var/lib/jenkins/secrets/initialAdminPassword
Apr 08 13:39:59 ip-172-31-43-52 jenkins[6209]: ****
Apr 08 13:39:59 ip-172-31-43-52 jenkins[6209]: ****
Apr 08 13:39:59 ip-172-31-43-52 jenkins[6209]: ****
Apr 08 13:40:48 ip-172-31-43-52 jenkins[6209]: 2024-04-08 13:40:48.062+0000 [id=31] INFO jenkins.InitReactorRunner$1#onAttained: Completed initialization
Apr 08 13:40:48 ip-172-31-43-52 jenkins[6209]: 2024-04-08 13:40:48.105+0000 [id=24] INFO hudson.lifecycle.Lifecycle#onReady: Jenkins is fully up and running
Apr 08 13:40:48 ip-172-31-43-52 system[1]: Started Jenkins Continuous Integration Server.
Apr 08 13:40:48 ip-172-31-43-52 jenkins[6209]: 2024-04-08 13:40:48.819+0000 [id=49] INFO h.m.DownloadService$Downloadable#load: Obtained the updated data
Apr 08 13:40:48 ip-172-31-43-52 jenkins[6209]: 2024-04-08 13:40:48.820+0000 [id=49] INFO hudson.util.Retrier#start: Performed the action check updates ser
lines 1-20/20 (END)
```

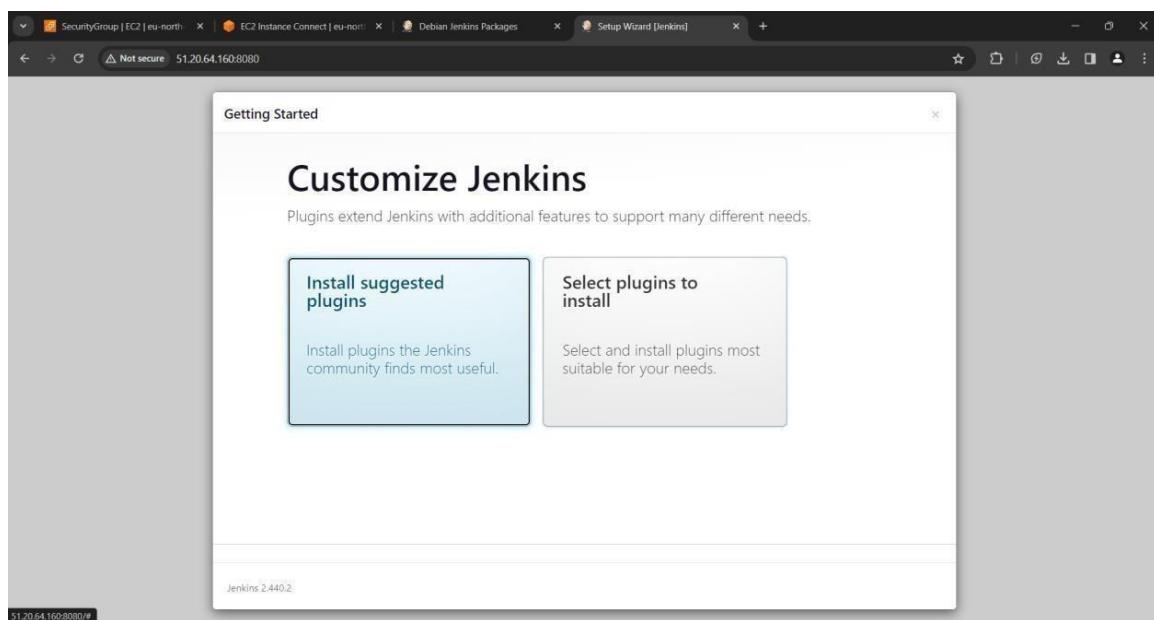
At the bottom of the terminal, it shows the IP address: "PublicIP: 51.20.64.160 PrivateIP: 172.31.45.52".

STEP 4: Configuring Jenkins

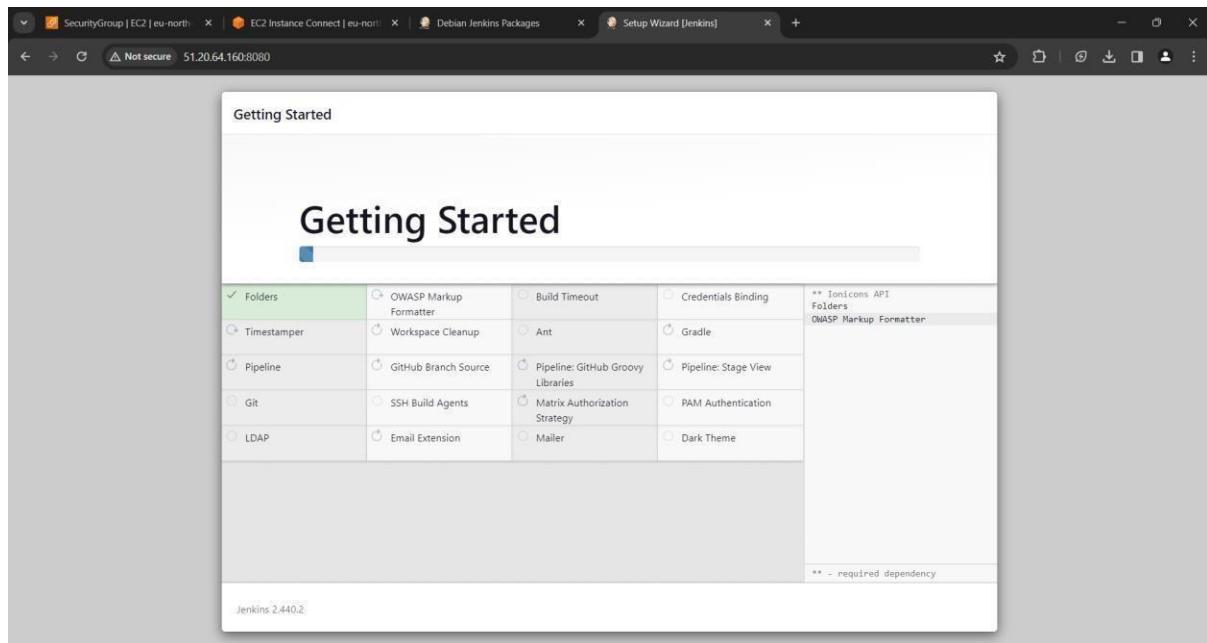
1. Jenkins will be initially locked. The password will be located at the location provided on the screen. Copy it.
2. Go back to the terminal and execute
\$ sudo cat /var/lib/Jenkins/secrets/initialAdminPassword
3. Copy the password from the terminal, paste it on the Jenkins webpage and press Continue.



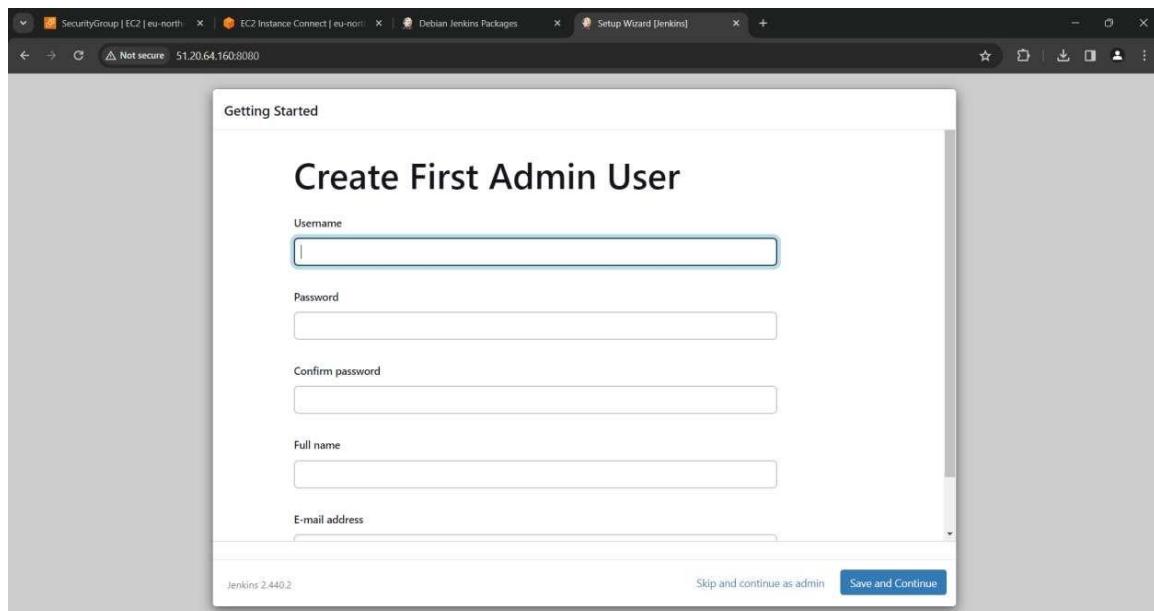
4. Select Installed suggested plugins to install the most used plugins in Jenkins.



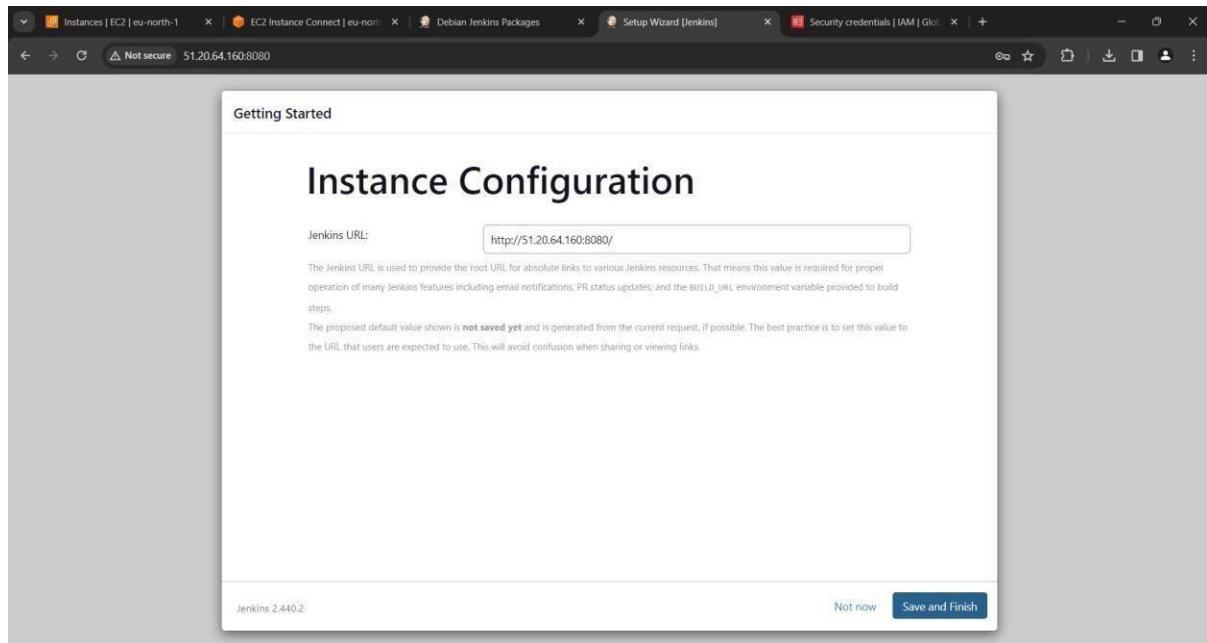
5. The installation will take some time to complete.



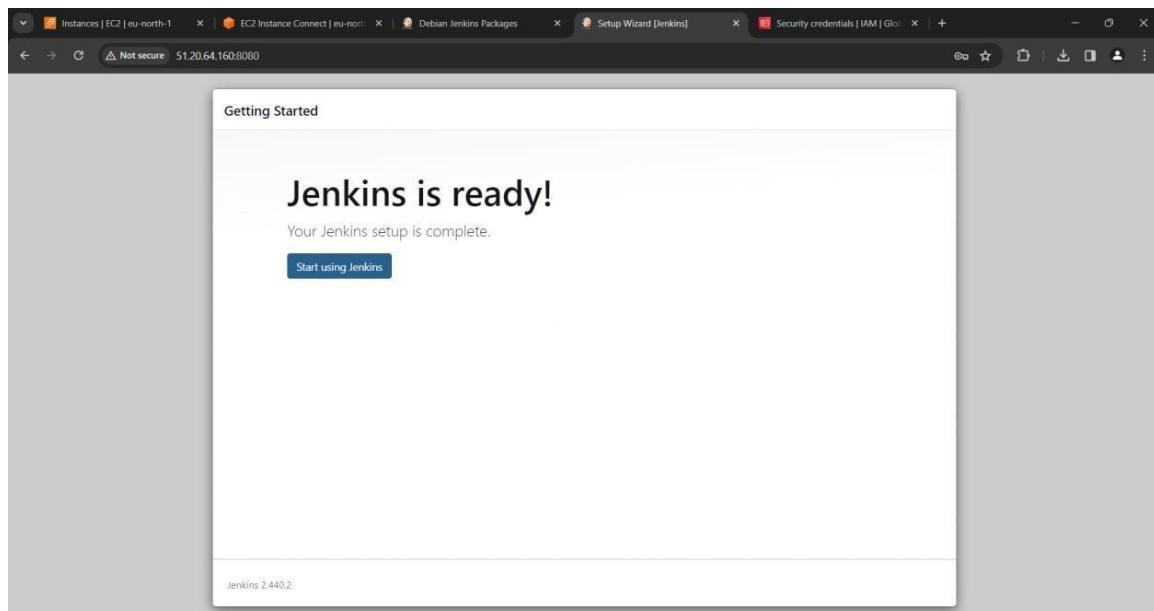
6. Create a user login on the next page. Provide the necessary information and press **Save and Continue**.



7. Check whether the Jenkins URL is the same as the one mentioned in the address bar and press **Save and Finish**.



8. Now Jenkins has been successfully configured on the cloud. Click on Start using Jenkins to work with it.



9. The Jenkins web page would look like below. Jobs can be built here.

The screenshot shows a browser window with multiple tabs open, including 'Instances | EC2 | eu-north-1', 'EC2 Instance Connect | eu-north-1', 'Debian Jenkins Packages', 'Dashboard [Jenkins]', and 'Security credentials | IAM | Global'. The main content area is the Jenkins dashboard, titled 'Welcome to Jenkins!'. It features a sidebar with links for 'New Item', 'People', 'Build History', 'Manage Jenkins', and 'My Views'. Below the sidebar are two sections: 'Build Queue' (empty) and 'Build Executor Status' (showing 1 idle and 2 idle executors). The main area has a heading 'Start building your software project' with three buttons: 'Create a job' (with a '+' icon), 'Set up a distributed build' (with a computer icon), and 'Set up an agent' (with a monitor icon). To the right of these are 'Configure a cloud' (with a cloud icon) and 'Learn more about distributed builds' (with a question mark icon). At the bottom right of the dashboard, it says 'REST API' and 'Jenkins 2.440.2'.

RESULT:

Thus, Jenkins has been installed on the cloud.

Ex. No: 4

CREATE CI PIPELINE USING JENKINS

Date:

AIM:

To create a CI pipeline using Jenkins.

PROCEDURE:

STEP 1: Click and open this GitHub repository (<https://github.com/benc-uk/python-demoapp.git>). And also click fork to make the changes in your own account.

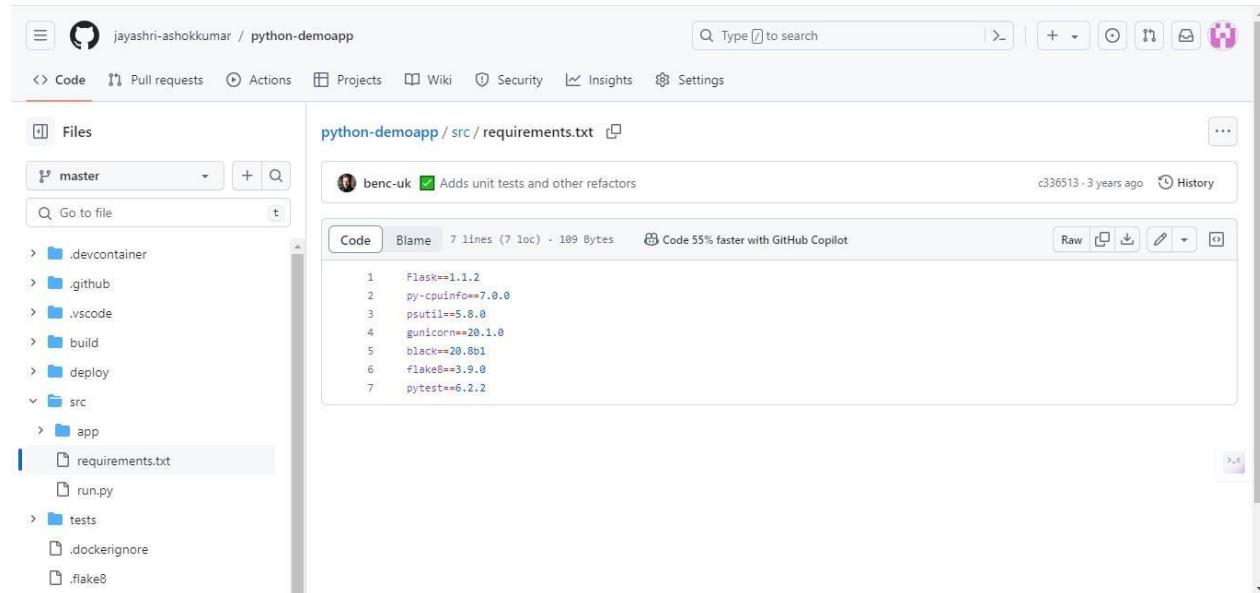
(Note: The username and password in GitHub, Jenkins and Docker Hub website should be the same.)

The screenshot shows the GitHub repository page for 'python-demoapp'. The repository has been archived by the owner on Nov 9, 2023, and is now read-only. It contains 4 commits from the 'master' branch, all made by 'benc-uk'. The commits include adding a devcontainer, force CI build and run, adding limiting in vscode, adding unit tests and other refactors, a 2021 refresh, and another force CI build and run. The repository has 880 forks and 119 stars. It is tagged as a public archive. The sidebar includes links for Readme, MIT license, Activity, 119 stars, 5 watching, 806 forks, and Report repository.

STEP 2: Now you have moved into your own GitHub account.

The screenshot shows the GitHub forked repository page for 'python-demoapp', which was forked from 'benc-uk/python-demoapp'. The repository is up-to-date with the master branch. It contains 4 commits from the 'master' branch, all made by 'benc-uk'. The commits are identical to those in the original repository. The repository has 0 stars, 0 forks, and 0 watching. It is tagged as a public archive. The sidebar includes links for Readme, MIT license, Activity, 0 stars, 0 watching, 0 forks, and Releases (4 tags).

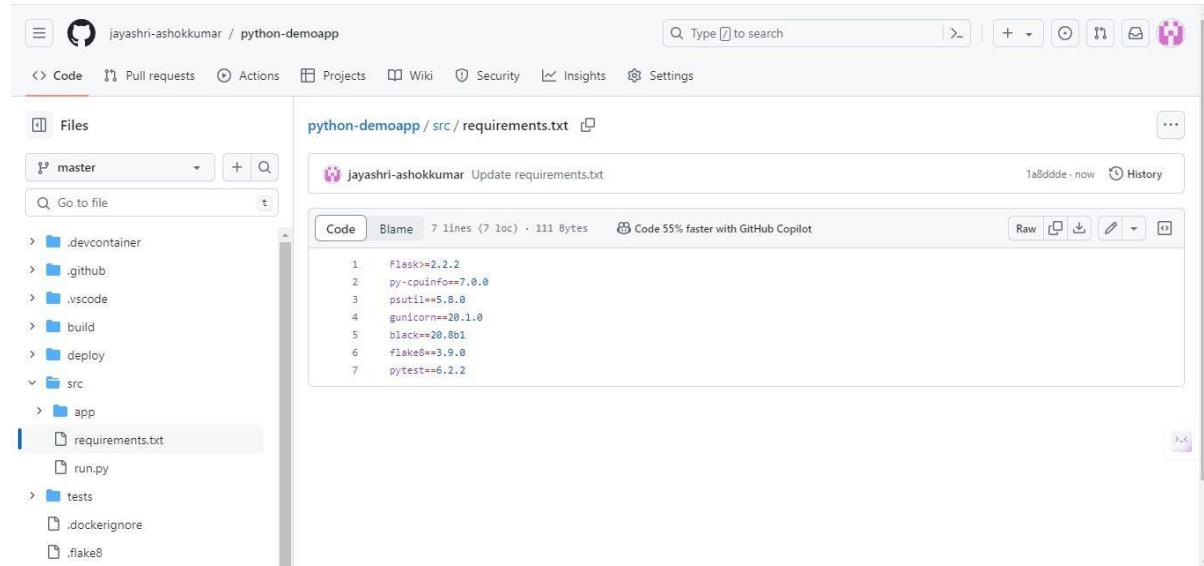
STEP 3: Click the src directory, choose requirements.txt and you will be able to see the below code.



A screenshot of a GitHub repository page for 'python-demoapp'. The left sidebar shows a tree view of the project structure: .devcontainer, .github, .vscode, build, deploy, src (which is expanded to show requirements.txt and run.py), app, tests, .dockerignore, and .flake8. The main pane displays the contents of 'src/requirements.txt'. The commit history shows a single entry by 'benc-uk' with the message 'Adds unit tests and other refactors', dated 'c336513 · 3 years ago'. The code itself is as follows:

```
1 Flask==1.1.2
2 py-cpuinfo==7.0.0
3 psutil==5.8.0
4 gunicorn==20.1.0
5 black==20.8b1
6 flake8==3.9.0
7 pytest==6.2.2
```

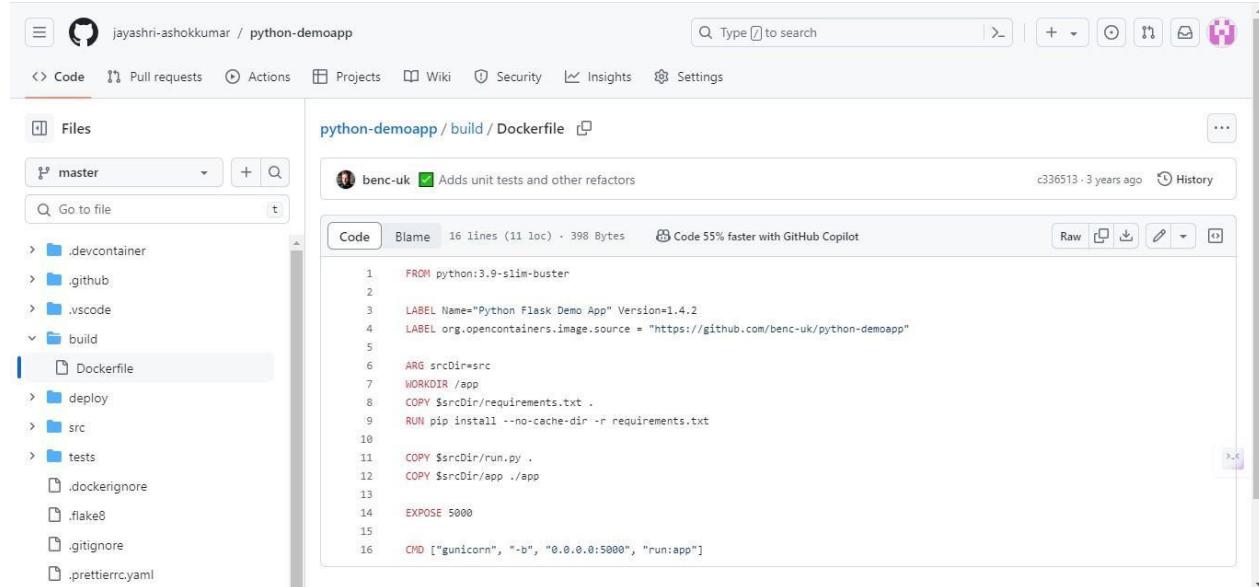
STEP 4: Now, you can make the below changes in the code.



A screenshot of the same GitHub repository page after changes have been made. The commit history now shows a new entry by 'jayashri-ashokkumar' with the message 'Update requirements.txt', dated '1a8ddde · now'. The code in the main pane has been updated to reflect these changes:

```
1 Flask>=2.2.2
2 py-cpuinfo>=7.0.0
3 psutil>=5.8.0
4 gunicorn>=20.1.0
5 black>=20.8b1
6 flake8>=3.9.0
7 pytest>=6.2.2
```

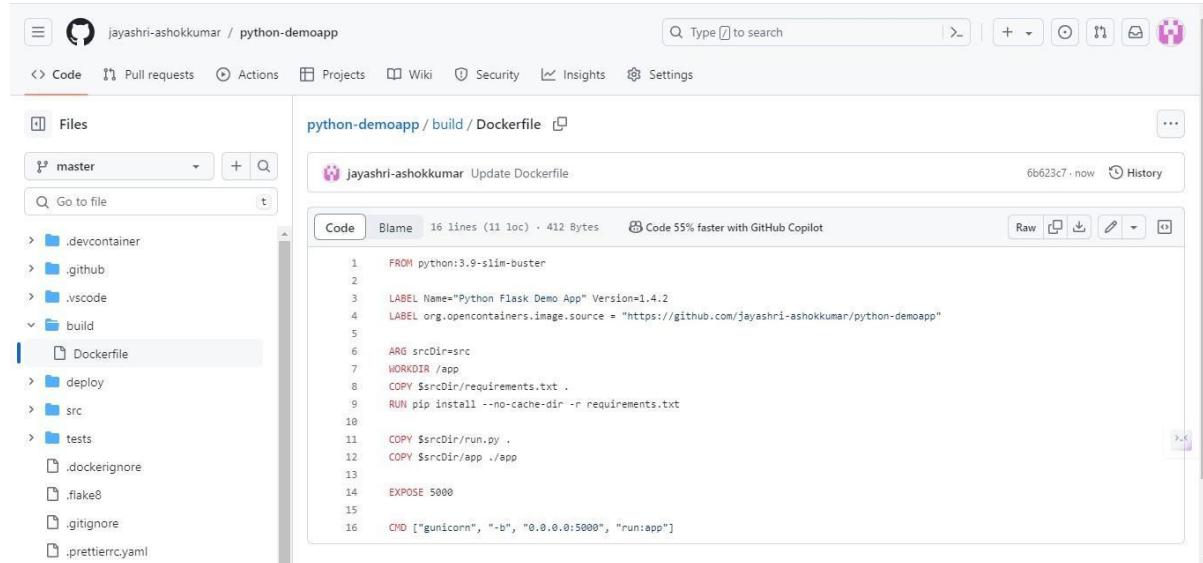
STEP 5: Click build directory and choose Dockerfile and you will be able to see the below code.



The screenshot shows a GitHub repository interface. The left sidebar lists files in the 'master' branch: .devcontainer, .github, .vscode, build (which is expanded to show Dockerfile, deploy, src, tests, .dockerignore, .flake8, .gitignore, and .prettierrc.yaml), and .prettierrc.yaml. The right pane displays the 'Dockerfile' in the 'build' directory. The file content is as follows:

```
FROM python:3.9-slim-buster
LABEL Name="Python Flask Demo App" Version=1.4.2
LABEL org.opencontainers.image.source = "https://github.com/benc-uk/python-demoapp"
ARG srcDir=src
WORKDIR /app
COPY $srcDir/requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY $srcDir/run.py .
COPY $srcDir/app ./app
EXPOSE 5000
CMD ["gunicorn", "-b", "0.0.0.0:5000", "run:app"]
```

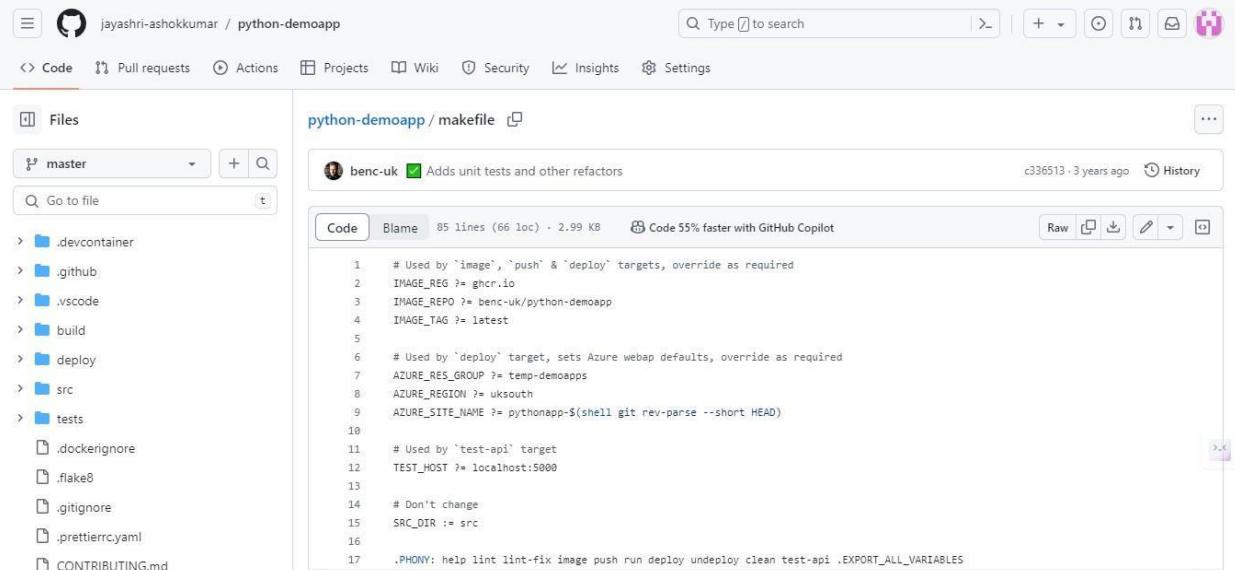
STEP 6: Replace the above link with your own GitHub repository link.



The screenshot shows the same GitHub repository interface as the previous one, but with a different URL in the Dockerfile's LABEL directive. The 'Dockerfile' content is now:

```
FROM python:3.9-slim-buster
LABEL Name="Python Flask Demo App" Version=1.4.2
LABEL org.opencontainers.image.source = "https://github.com/jayashri-ashokkumar/python-demoapp"
ARG srcDir=src
WORKDIR /app
COPY $srcDir/requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY $srcDir/run.py .
COPY $srcDir/app ./app
EXPOSE 5000
CMD ["gunicorn", "-b", "0.0.0.0:5000", "run:app"]
```

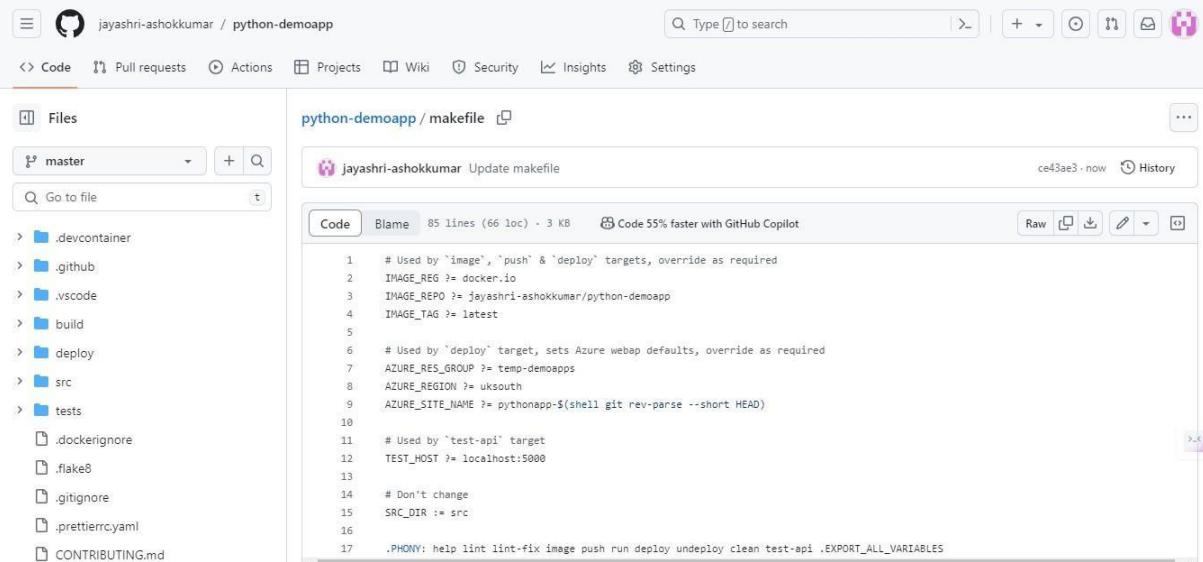
STEP 7: Click on makefile, and you will be able to see the below code.



The screenshot shows a GitHub repository page for 'python-demoapp'. The 'Code' tab is selected, and the 'makefile' file is open. The code content is as follows:

```
1 # Used by `image`, `push` & `deploy` targets, override as required
2 IMAGE_REG ?= ghcr.io
3 IMAGE_REPO ?= benc-uk/python-demoapp
4 IMAGE_TAG ?= latest
5
6 # Used by `deploy` target, sets Azure webapp defaults, override as required
7 AZURE_RES_GROUP ?= temp-demoapps
8 AZURE_REGION ?= ukouth
9 AZURE_SITE_NAME ?= pythonapp-$shell git rev-parse --short HEAD)
10
11 # Used by `test-api` target
12 TEST_HOST ?= localhost:5000
13
14 # Don't change
15 SRC_DIR := src
16
17 .PHONY: help lint lint-fix image push run deploy undeploy clean test-api .EXPORT_ALL_VARIABLES
```

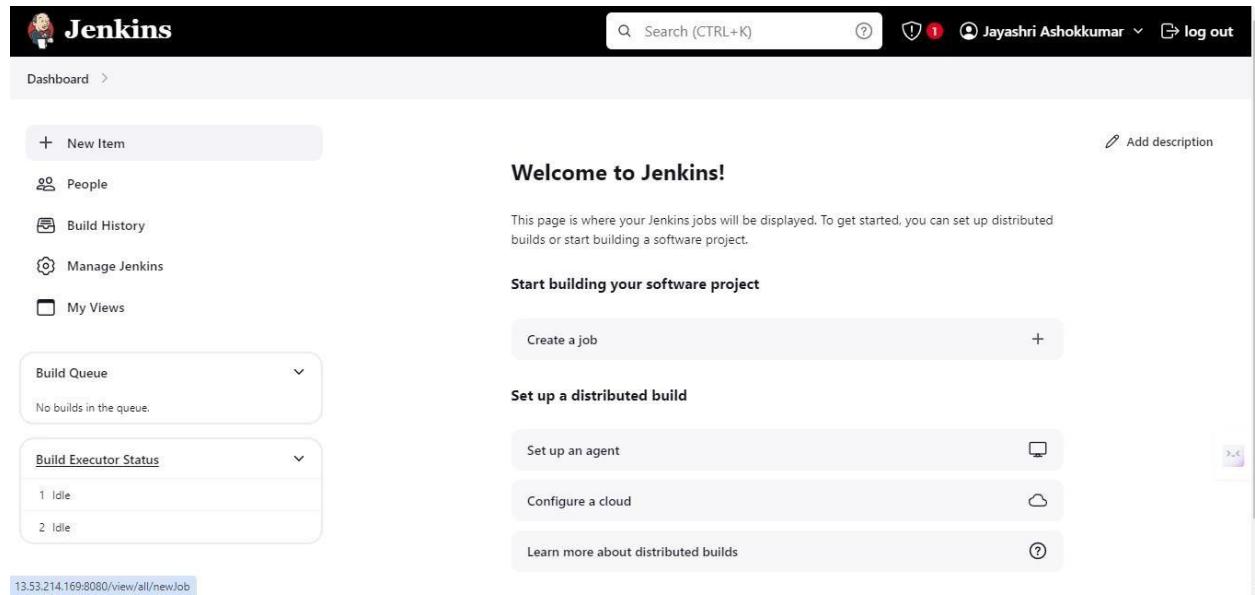
STEP 8: Replace the above name in the code with your own GitHub name and also replace the ghcr.io with docker.io.



The screenshot shows the same GitHub repository page for 'python-demoapp'. The 'Code' tab is selected, and the 'makefile' file is open. The code content has been modified to reflect the user's GitHub handle and Docker registry:

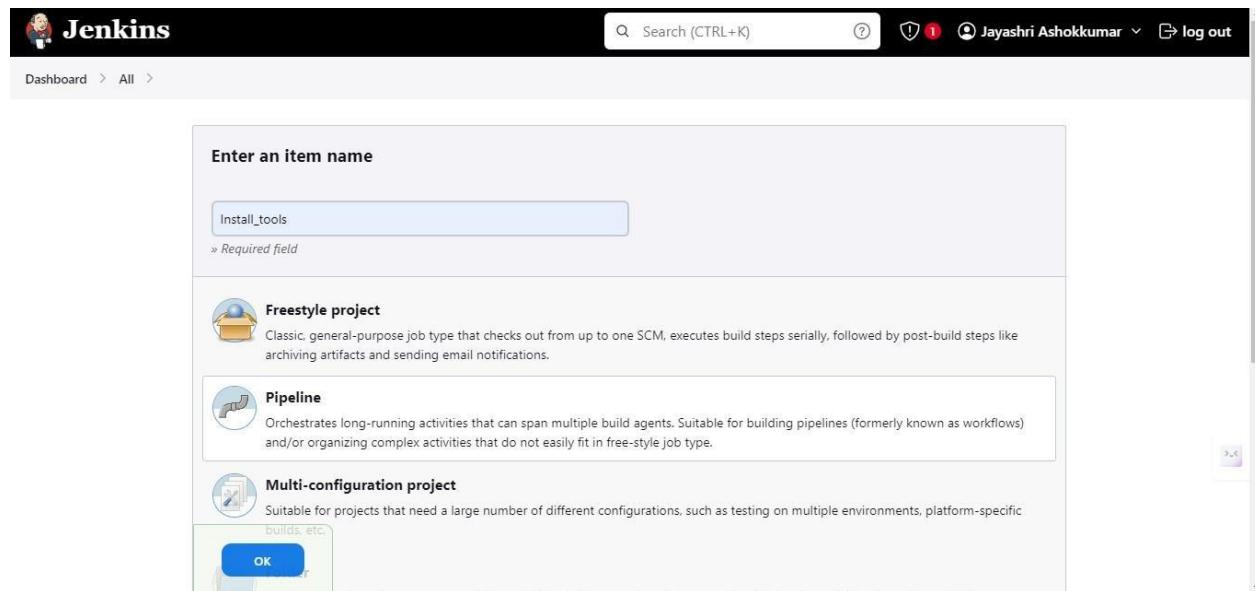
```
1 # Used by `image`, `push` & `deploy` targets, override as required
2 IMAGE_REG ?= docker.io
3 IMAGE_REPO ?= jayashri-ashokkumar/python-demoapp
4 IMAGE_TAG ?= latest
5
6 # Used by `deploy` target, sets Azure webapp defaults, override as required
7 AZURE_RES_GROUP ?= temp-demoapps
8 AZURE_REGION ?= ukouth
9 AZURE_SITE_NAME ?= pythonapp-$shell git rev-parse --short HEAD)
10
11 # Used by `test-api` target
12 TEST_HOST ?= localhost:5000
13
14 # Don't change
15 SRC_DIR := src
16
17 .PHONY: help lint lint-fix image push run deploy undeploy clean test-api .EXPORT_ALL_VARIABLES
```

STEP 9: Open the Jenkins page, and click on the new item that is present in the dashboard.



The screenshot shows the Jenkins dashboard. On the left sidebar, there is a button labeled '+ New Item'. The main area features a 'Welcome to Jenkins!' message and a 'Start building your software project' section. Below this, there are sections for 'Set up a distributed build', 'Set up an agent', 'Configure a cloud', and 'Learn more about distributed builds'. The URL at the bottom of the page is 13.53.214.169:8080/view/all/newJob.

STEP 10: Now Enter the item name and select pipeline, then click on OK.



The screenshot shows a 'Enter an item name' dialog box. In the input field, the text 'Install_tools' is entered. Below the input field, there are three options: 'Freestyle project', 'Pipeline', and 'Multi-configuration project'. The 'Pipeline' option is selected, and its description is visible. At the bottom of the dialog box, there is an 'OK' button.

STEP 11: Click > Install_tools > Configure > Pipeline. In the definition section of Pipeline, Choose Pipeline script and the script area will be visible in which you will type the below code.

```

pipeline {
    agent any
    stages {
        stage('Python Version') {
            steps {
                sh "sudo apt update"
                sh "sudo apt -y upgrade"
                sh "sudo apt install python3.10-venv -y"
                sh "python3 -v"
            }
        }
        stage('Docker Version'){
            steps{
                sh "docker -v"
            }
        }
        stage('Install Make'){
            steps{
                sh "sudo apt install make -y"
                sh "sudo apt install make-guile -y"
            }
        }
    }
}

```

Now, Click on Save.

The screenshot shows the Jenkins Pipeline configuration interface. The top navigation bar includes 'Dashboard', 'Install_tools', and 'Configuration'. The left sidebar has tabs for 'General', 'Advanced Project Options', and 'Pipeline', with 'Pipeline' currently selected. The main area is titled 'Configure Pipeline' and contains a 'Definition' dropdown set to 'Pipeline script'. Below it is a code editor containing the pipeline script shown above. A checkbox labeled 'Use Groovy Sandbox' is checked. At the bottom are 'Save' and 'Apply' buttons.

```

1> pipeline {
2>     agent any
3>
4>     stages {
5>         stage('Python Version') {
6>             steps {
7>                 sh "sudo apt update"
8>                 sh "sudo apt -y upgrade"
9>                 sh "sudo apt install python3.10-venv -y"
10>                sh "python3 -v"
11>            }
12>        }
13>        stage('Docker Version'){
14>            steps{
15>                sh "docker -v"
16>            }
17>        }
18>    }
19>

```

STEP 12: Click on the Install_tools and select the build now.

The screenshot shows the Jenkins dashboard with the 'Install_tools' pipeline selected. A context menu is open over the pipeline name, with 'Build Now' highlighted. Other options in the menu include 'Changes', 'Configure', 'Delete Pipeline', 'Full Stage View', 'Rename', and 'Pipeline Syntax'. To the right of the pipeline name, there are buttons for 'Add description' and 'Disable Project'. Below the pipeline name, a message says 'No data available. This Pipeline has not yet run.' On the left, there's a sidebar with links like 'Status', 'Changes', 'Build Now', 'Configure', 'Delete Pipeline', 'Full Stage View', 'Rename', and 'Pipeline Syntax'. At the bottom, there's a 'Permalinks' section and a 'Build History' card that says 'No builds'.

STEP 13: In the Build history section, click on the builds then you will able to see the console output as 'FINISHED: SUCCESS'. Now The Pipeline is built.

The screenshot shows the Jenkins dashboard with the 'Install_tools' pipeline selected. The 'Console Output' tab is active, displaying the build logs. The logs show the pipeline starting, running on Jenkins, and executing a stage that includes installing Python and updating packages. It also includes a warning about using apt in scripts and a list of package downloads. The sidebar on the left shows other tabs like 'Status', 'Changes', 'Console Output' (which is selected), 'View as plain text', 'Edit Build Information', 'Delete build', 'Restart from Stage', 'Replay', 'Pipeline Steps', and 'Workspaces'. The top right shows user information and log out options.

The screenshot shows the Jenkins Dashboard. In the top right corner, there is a "New Item" button and an "Add description" link. Below these are links for "People", "Build History", "Manage Jenkins", and "My Views". A "Docker Swarm Dashboard" section is displayed, featuring a table with columns: S (Status), W (Workload), Name, Last Success, Last Failure, and Last Duration. One entry is shown: "Installtools" with a green checkmark icon, a cloud icon, and a duration of "2 hr 17 min #4". Below the table are "Icon legend" and three "Atom feed" links: "Atom feed for all", "Atom feed for failures", and "Atom feed for just latest builds". On the left, there are two expandable sections: "Build Queue" (No builds in the queue) and "Build Executor Status" (1 Idle, 2 Idle).

STEP 14: Click > Dashboard > Manage Jenkins > Plugins. In Plugins, Select Available Plugins. Here, search for docker and select the associated plugins (Docker, docker pipeline, docker build step, CloudBees Docker Build and Publish) then install it.

The screenshot shows the Jenkins Manage Jenkins > Plugins page. The search bar at the top contains the text "docker". On the left, there is a sidebar with options: "Updates" (selected), "Available plugins" (highlighted in blue), "Installed plugins", "Advanced settings", and "Download progress". The main area displays a table of available plugins. The first plugin listed is "CloudBees Docker Build and Publish" version 1.4.0, released "1 yr 7 mo ago". Its description states: "This plugin enables building Dockerfile based projects, as well as publishing of the built images/repos to the docker registry." The second plugin is "Amazon ECR" version 1.114.vfd22430621f5, released "1 yr 2 mo ago". Its description states: "This plugin generates Docker authentication token from Amazon Credentials to access Amazon ECR." The third plugin is "JFrog" version 1.5.0, released "7 mo 20 days ago". Its description states: "The Jenkins JFrog Plugin allows for easy integration between Jenkins and the JFrog Platform. This integration allows your build jobs to deploy artifacts and resolve dependencies to and from Artifactory, and then have them linked to the build job that created them. It also allows you to scan your artifacts and builds with JFrog Xray and distribute your software package to remote".

STEP 15: In Download progress, you will be able to see the set of plugins that were installed.

The screenshot shows the Jenkins 'Manage Jenkins' section with the 'Plugins' tab selected. On the left, there's a sidebar with links: 'Updates', 'Available plugins', 'Installed plugins', 'Advanced settings', and 'Download progress'. The 'Download progress' link is highlighted with a grey background. The main area lists 15 plugins with their status as 'Success' indicated by green checkmarks. The list includes: Cloud Statistics, Authentication Tokens API, Docker Commons, Apache HttpComponents Client 5.x API, Docker API, Docker, Loading plugin extensions, Docker Pipeline, Javadoc, JSch dependency, Maven Integration, docker-build-step, SSH server, CloudBees Docker Build and Publish, and Loading plugin extensions. At the bottom, there are two links: 'Go back to the top page' and 'Restart Jenkins when installation is complete and no jobs are running'.

Plugin	Status
Cloud Statistics	Success
Authentication Tokens API	Success
Docker Commons	Success
Apache HttpComponents Client 5.x API	Success
Docker API	Success
Docker	Success
Loading plugin extensions	Success
Docker Pipeline	Success
Javadoc	Success
JSch dependency	Success
Maven Integration	Success
docker-build-step	Success
SSH server	Success
CloudBees Docker Build and Publish	Success
Loading plugin extensions	Success

STEP 16: Run the following commands in the AWS ubuntu terminal.

1. To Create the Docker Group:

```
$sudo groupadd docker
```

2. To Add the Jenkins User to the Docker Group:

```
$sudo usermod -aG docker jenkins
```

(Replace 'jenkins' with the actual username of the Jenkins user)

3. To Verify the Group Membership:

```
$groups Jenkins
```

4. To install the docker:

```
$sudo apt install docker.io
```

5. To restart jenkins:

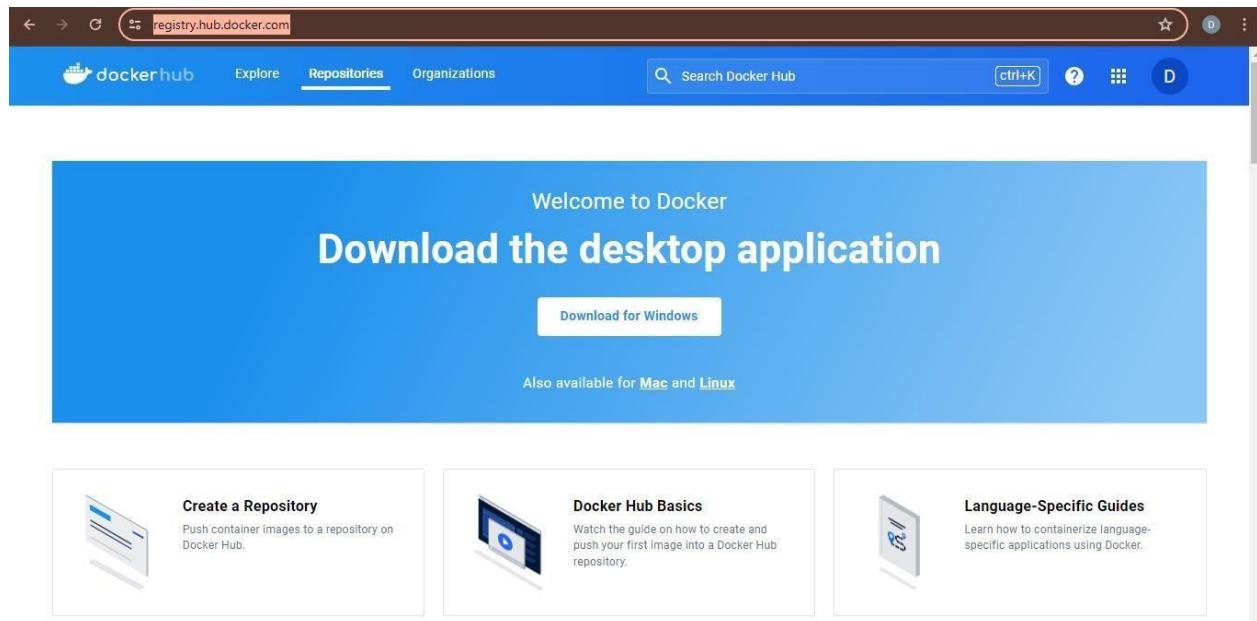
```
$sudo systemctl restart Jenkins
```

STEP 17: Click > Manage Jenkins > Tools. Scroll for the docker section and perform the following actions in the image.

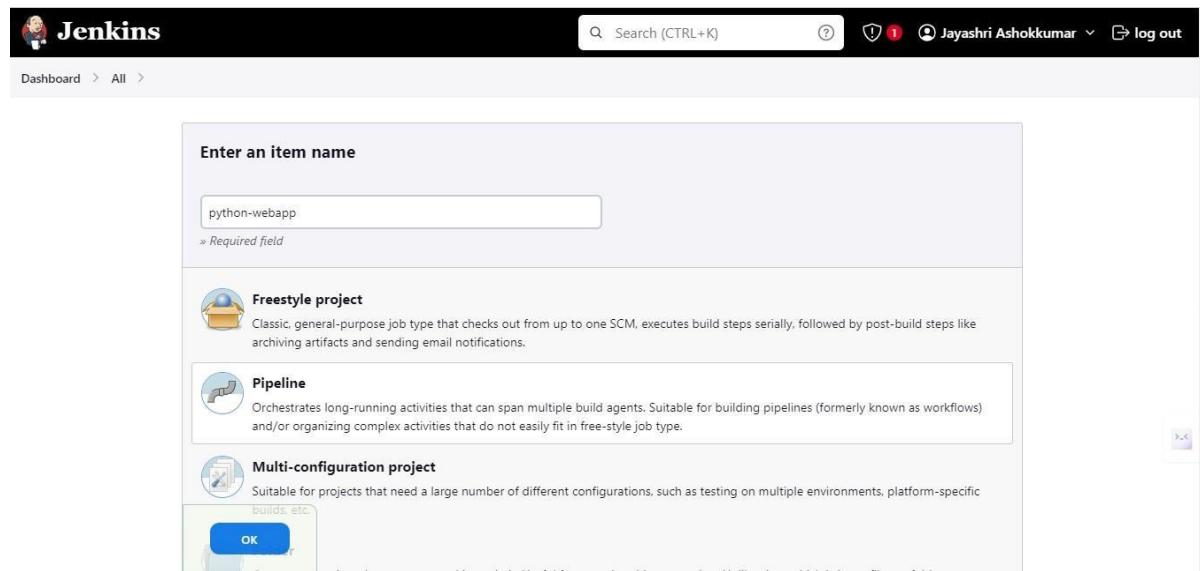
The screenshot shows the Jenkins Manage Jenkins > Tools page. Under the Docker section, a configuration for 'docker' is displayed. The 'Name' field contains 'docker'. The 'Install automatically' checkbox is checked. A nested configuration for 'Download from docker.com' is shown, with the 'Docker version' field set to 'latest'. There is also an 'Add Installer' dropdown. At the bottom, there are 'Save' and 'Apply' buttons.

STEP 18: Open Browser and search for docker hub, then create an account in docker hub.

The screenshot shows a web browser window with the URL 'hub.docker.com/signup'. The page title is 'Create your account'. It features two sign-in options: 'Continue with Google' and 'Continue with GitHub', each with its respective logo. Below these is a 'OR' link. There are input fields for 'Email' and 'Username', both with placeholder text. A note below the email field says 'We suggest signing up with your work email address.' At the bottom is a 'Password' input field with a visibility toggle icon.



STEP 19: Click on the new item in the Jenkins dashboard. Now Enter the item name and select pipeline, then click on OK.



STEP 20: Click > dashboard > python-webapp > configure. In the general section, select ‘Discard old builds’ and set the max of builds to 1.

The screenshot shows the Jenkins configuration interface for a pipeline named 'python_webapp'. The 'General' tab is active. Under 'Discard old builds', the 'Log Rotation' strategy is selected. The 'Days to keep builds' field is empty. The 'Max # of builds to keep' field contains the value '1'. At the bottom, there are 'Save' and 'Apply' buttons.

STEP 21: Scroll down for Pipeline. In the Script area, run the below code.

```
pipeline {
    agent any
    stages {
        stage('Git Checkout') {
            steps {
                git changelog: false, poll: false, url:''
            }
        }
        stage('Docker Build') {
            steps {
                sh "make image"
            }
        }
        stage('Docker Push') {
            steps {
                script{
                    withDockerRegistry(credentialsId: ' ', toolName: ' ') {
                        sh "make push"
                    }
                }
            }
        }
    }
}
```

The screenshot shows the Jenkins Pipeline configuration page for a project named 'python_webapp'. The 'Pipeline' tab is selected. The 'Definition' section contains a 'Pipeline script' input field with the following Groovy code:

```

1 pipeline {
2     agent any
3
4     stages {
5         stage('Git Checkout') {
6             steps {
7                 git changelog: false, poll: false, url: 'https://github.com/dk172923/python-webapp.git'
8             }
9         }
10        stage('Docker Build') {
11            steps {
12                sh "make image"
13            }
14        }
15        stage('Docker Push') {
16            steps {
17                dockerPush()
18            }
19        }
20    }
21 }

```

Below the script, there is a checkbox labeled 'Use Groovy Sandbox' which is checked. At the bottom are 'Save' and 'Apply' buttons.

STEP 22: Click on pipeline syntax and Select the Snippet generator

1. In the Sample step, choose the git: Git.

The screenshot shows the 'Pipeline Syntax' Snippet Generator interface. The 'Snippet Generator' tab is selected. On the left, a sidebar lists options: Back, Snippet Generator (selected), Declarative Directive Generator, Declarative Online Documentation, Steps Reference, Global Variables Reference, Online Documentation, Examples Reference, and IntelliJ IDEA GDSL. The main area is titled 'Overview' and contains a description of the Snippet Generator. Below it is a 'Steps' section with a 'Sample Step' dropdown set to 'git: Git'. Underneath are fields for 'Repository URL' containing 'https://github.com/Dhakshineswari/python-demoapp.git' and 'Branch' containing 'main'.

2. Paste your GitHub URL in the URL section.
3. Type as 'main' in Branch and deselect the options.

Dashboard > python_webapp > Pipeline Syntax

git ?

Repository URL ?

Branch ?

Credentials ?

+ Add ▾

Include in polling? ?

Include in changelog? ?

Generate Pipeline Script

4. Click on Generate pipeline script.

Dashboard > python_webapp > Pipeline Syntax

git branch: 'main', changelog: false, poll: false, url: 'https://github.com/Dhakshineswari/python-demoapp.git'

Global Variables

There are many features of the Pipeline that are not steps. These are often exposed via global variables, which are not supported by the snippet generator. See the [Global Variables Reference](#) for details.

Jenkins 2.440.3

STEP 23: In Step 21, paste the above generator pipeline script in the URL portion of the code.

The screenshot shows the Jenkins Pipeline configuration page. The pipeline script is defined as follows:

```

1 pipeline {
2     agent any
3
4     stages {
5         stage('Git Checkout') {
6             steps {
7                 git branch: 'main', changelog: false, poll: false, url: 'https://github.com/Dhakshineswari/python-demoapp'
8             }
9         }
10        stage('Docker Build') {
11            steps {
12                sh "make image"
13            }
14        }
15    }
16}

```

Below the script, there is a checkbox for "Use Groovy Sandbox". At the bottom are "Save" and "Apply" buttons.

STEP 24: Again Click on pipeline and select the Snippet generator.

1. In the Sample step, choose the below option.

The screenshot shows the Jenkins Snippet Generator interface. The "Steps" section is expanded, showing the "withDockerRegistry" step. The configuration fields are:

- withDockerRegistry: Sets up Docker registry endpoint
- withDockerRegistry ?
- Docker registry URL ?
- Registry credentials

2. Make the Registry Credentials null and click Jenkins.

The screenshot shows the Jenkins Pipeline Syntax configuration page for a pipeline named 'python-webapp'. On the left, there's a sidebar with links to Global Variables Reference, Online Documentation, Examples Reference, and IntelliJ IDEA GDSL. The main area displays a 'Sample Step' section for 'withDockerRegistry'. This step has a 'Docker registry URL' field (empty), a 'Registry credentials' dropdown set to '- none -', and a 'Jenkins Credentials Provider' dropdown showing '(Default)'. A 'Generate Pipeline Script' button is at the bottom.

3. Now, provide the username and password (GitHub) and click save.

The screenshot shows a 'Jenkins Credentials Provider: Jenkins' dialog box. It contains fields for 'Username' (set to 'jayashri_ashokkumar'), 'Password' (containing '*****'), 'ID' (empty), and 'Description' (set to 'docker-cred'). There are 'Cancel' and 'Add' buttons at the bottom. The background shows the same Jenkins Pipeline Syntax configuration page as the previous screenshot.

4. The Credentials will have been added in the Registry Credentials section select the docker below, then click on Generate pipeline script.

The screenshot shows the 'Pipeline Syntax' configuration screen in IntelliJ IDEA GDSL. It includes fields for 'Docker registry URL', 'Registry credentials' (set to 'jayashri_ashokkumar/******** (docker-cred)'), 'Docker installation' (set to 'docker'), and a 'Generate Pipeline Script' button. Below the button is a code editor containing Groovy script code:

```
// This step should not normally be used in your script. Consult the inline help for details.  
withDockerRegistry(credentialsId: 'f3fed2f0-5a1d-4494-926b-14fb90e5a36c', toolName: 'docker') {  
    // some block  
}
```

STEP 25: In Step 21, paste the above generator pipeline script in the Credentials ID and tool name portion of the code.

The screenshot shows the 'Pipeline' configuration screen in Jenkins. The 'Definition' dropdown is set to 'Pipeline script'. The main area contains a code editor with Groovy script code. The 'Pipeline' tab is selected in the left sidebar. A checkbox for 'Use Groovy Sandbox' is checked. At the bottom are 'Save' and 'Apply' buttons.

```
13 *      sh "make image"  
14 }  
15 }  
16 }  
17 * stage('Docker Push') {  
18 *     steps {  
19 *         script{  
20 *             withDockerRegistry(credentialsId: 'f3fed2f0-5a1d-4494-926b-14fb90e5a36c', toolName: 'docker') {  
21 *                 sh "make push"  
22 *             }  
23 *         }  
24 *     }  
25 }  
26 }  
27 }  
28 }
```

STEP 26: Click > dashboard > python-webapp > build now.

The screenshot shows the Jenkins dashboard for the 'python-webapp' project. The top navigation bar includes 'Search (CTRL+K)', a user profile for 'Jayashri Ashokkumar', and 'log out'. The left sidebar contains links for 'Status', 'Changes', 'Build Now', 'Configure', 'Delete Pipeline', 'Full Stage View', 'Rename', and 'Pipeline Syntax'. A 'Build History' section shows the last build (#25) from April 18, 2024, at 14:31, with 'No Changes'. The main area features a 'Stage View' chart with three stages: 'Git Checkout' (1s), 'Docker Build' (19s), and 'Docker Push' (21s). Below the chart is a 'Permalinks' section with a link to the last build. The bottom of the page has a footer with links for 'Atom feed for all', 'Atom feed for failures', and 'Atom feed for just latest builds'.

STEP 27: Now CI Pipeline has been built Successfully.

The screenshot shows the Jenkins dashboard with two projects listed: 'Install_tools' and 'python-webapp'. Both projects show a green checkmark icon, indicating they have been successfully built. The 'python-webapp' project was last successful 45 seconds ago. The dashboard also displays the 'Build Queue' (empty), 'Build Executor Status' (one build in progress for 'python-webapp'), and various Jenkins management links like 'New Item', 'People', 'Manage Jenkins', and 'My Views'.

RESULT:

Thus, the CI Pipeline has been successfully created.

Ex. No: 5 CREATE A CD PIPELINE IN JENKINS AND DEPLOY IN CLOUD

Date:

AIM:

To create a CD pipeline in Jenkins and deploy it in the Cloud.

PROCEDURE:

STEP 1: Click Repositories > pythondemoapp > General. There, you will be able to see a command under Docker commands. Copy that command.

The screenshot shows the Docker Hub interface for a repository named `jayashriashokkumar28112003/python-demoapp`. In the `General` tab, under the `Docker commands` section, there is a text input field containing the command `docker push jayashriashokkumar28112003/python-demoapp: tagname`. Below this, the `Tags` section lists one tag: `latest`. The `Automated Builds` section provides instructions for setting up automated builds by connecting to GitHub or Bitbucket.

STEP 2: Now, go to your GitHub account. Scroll down to Containers and copy the command again for future use.

The screenshot shows a GitHub repository page with the `Containers` section visible. It includes a command to run the app in a container (`docker run --rm -it -p 5000:5000 ghcr.io/benc-uk/python-demoapp:latest`) and instructions for building a custom container using `make image`.

STEP 3: Now add the code (combination of the above commands) below to the code through which the CI Pipeline was built.

```
stage('Docker Deploy') {  
    steps {  
        script{  
            withDockerRegistry(credentialsId: 'dbc54abf-de2b-42b8-a0d2-f5539c9b8997',  
            toolName: 'docker') {  
                sh "docker images"  
                sh "docker run -d --rm -it -p 80:5000 jayashriashokkumar28112003/python-  
                webapp:latest"  
            }  
        }  
    }  
}
```

The screenshot shows the Jenkins Pipeline Configuration screen for a project named 'python-webapp'. The 'Pipeline' tab is selected. The pipeline script is displayed in a code editor:

```
Dashboard > python-webapp > Configuration  
  
Configure ?  
General  
Advanced Project Options  
Pipeline  
  
Script ?  
14 *  
15 }  
16 }  
17 }  
18 }  
19 }  
20 }  
21 }  
22 }  
23 }  
24 }  
25 }  
26 }  
27 }  
28 }  
29 }  
30 }  
31 }  
32 }  
33 }  
34 }  
35 }  
36 }  
37 }
```

The script defines a stage named 'Docker Deploy' with steps to push a Docker image and run it on port 80. A 'Use Groovy Sandbox' checkbox is checked. At the bottom, there are 'Save' and 'Apply' buttons, and a green 'Saved' message is displayed.

STEP 4: Click on Dashboard > python-webapp > Build Now

The screenshot shows the Jenkins dashboard for the 'python-webapp' project. On the left, there's a sidebar with options like Status, Changes, Build Now, Configure, Delete Pipeline, Full Stage View, Rename, Pipeline Syntax, and a Build History section showing two recent builds (#26 and #25) with no changes. The main area is titled 'Stage View' and displays a grid of four stages: Git Checkout, Docker Build, Docker Push, and Docker Deploy. Each stage has an average time listed below it: 1s, 9s, 13s, and 3s respectively. The Docker Build stage is highlighted in blue.

STEP 5: To access the Python web app copy and paste the public IP of the AWS instance and include port 5000 at the end of it. (For example: 41.204.111.57:5000)

The screenshot shows a browser window displaying the 'Python & Flask Demo App'. The title bar shows the URL as 'Not secure | 43.204.111.57:5000'. The main content area features a flask icon and the title 'Python & Flask Demo App'. Below the title, there is a brief description: 'This is a simple web application written in Python and using Flask. It has been designed with cloud demos & containers in mind. Demonstrating capabilities such as auto scaling, deployment to Azure or Kubernetes, or anytime you want something quick and lightweight to run & deploy.' There are three buttons at the bottom of this section: 'GitHub Project', 'Docker Images', and 'Get started with Azure & Python'. At the very bottom of the page, it says 'Microsoft ❤️ Open Source'.

RESULT:

Thus, the CD pipeline has been successfully created in Jenkins and deployed in the Cloud.

Ex. No: 6 **CREATE AN ANSIBLE PLAYBOOK FOR A SIMPLE WEB APPLICATION INFRASTRUCTURE**

AIM

To Create an Ansible Playbook for a simple web application infrastructure.

PROCEDURE

STEP 1: Create a directory named **ansible-practice1** and then navigate into that directory with the cd command.

The following commands are used to create a directory,

```
$ cd ~  
$ mkdir ansible-practice1  
$ cd ansible-practice1
```

To add content to playbook use the following commands,

```
$ nano playbook01.yml
```

```
ubuntu@ip-172-31-43-52:~/ansible-practice$ cd  
ubuntu@ip-172-31-43-52:~$ mkdir ansible-practice1  
ubuntu@ip-172-31-43-52:~$ cd ansible-practice1  
ubuntu@ip-172-31-43-52:~/ansible-practice1$ nano playbook01.yml  
ubuntu@ip-172-31-43-52:~/ansible-practice1$ █
```

STEP 2: Now create a playbook that contains the following content to be executed. The content to be added in the playbook (**playbook01.yml**),

```
---  
- hosts: localhost  
  tasks:  
    - name: Print message  
      debug:  
        msg: Hello Ansible World
```

NOTE: The host name in the content must be changed to localhost.

```

GNU nano 6.2                               playbook01.yml
---
- hosts: localhost
  tasks:
    - name: Print message
      debug:
        msg: Hello Ansible World

[ Wrote 6 lines ]
^G Help          ^O Write Out     ^W Where Is      ^X Cut           ^T Execute       ^C Location      ^-U Undo
^X Exit         ^R Read File     ^N Replace       ^U Paste          ^J Justify       ^/ Go To Line   ^-R Redo

```

STEP 3: Run **ansible-playbook** with the same connection arguments, use an inventory file named **inventory** and the **sammy** user to connect to the remote server. The command to run the ansible playbook,

```
$ ansible-playbook -i inventory playbook-01.yml -u sammy
```

```

ubuntu@ip-172-31-43-52:~/ansible-practice1$ ansible-playbook -i inventory playbook01.yml -u sammy
[WARNING]: Unable to parse /home/ubuntu/ansible-practice1/inventory as an inventory source
[WARNING]: No inventory was parsed, only implicit localhost is available
[WARNING]: provided hosts list is empty, only localhost is available. Note that the implicit localhost does not match 'all'

PLAY [localhost] ****
TASK [Gathering Facts] ****
ok: [localhost]

TASK [Print message] ****
ok: [localhost] => {
    "msg": "Hello Ansible World"
}

PLAY RECAP ****
localhost                  : ok=2    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

ubuntu@ip-172-31-43-52:~/ansible-practice1$ 

```

RESULT:

Thus, creating and executing an ansible playbook for a simple web application infrastructure has been executed successfully.

Ex. No: 7

BUILD A SIMPLE APPLICATION USING GRADLE

Date:

AIM

To build a simple application using Gradle.

PROCEDURE

Step 1: Create directory: `java_application`.

Step 2: Change to directory: `java_application`.

Step 3: Run `gradle init` command.

Step 4: Select project type (application).

Step 5: Choose implementation language (Java).

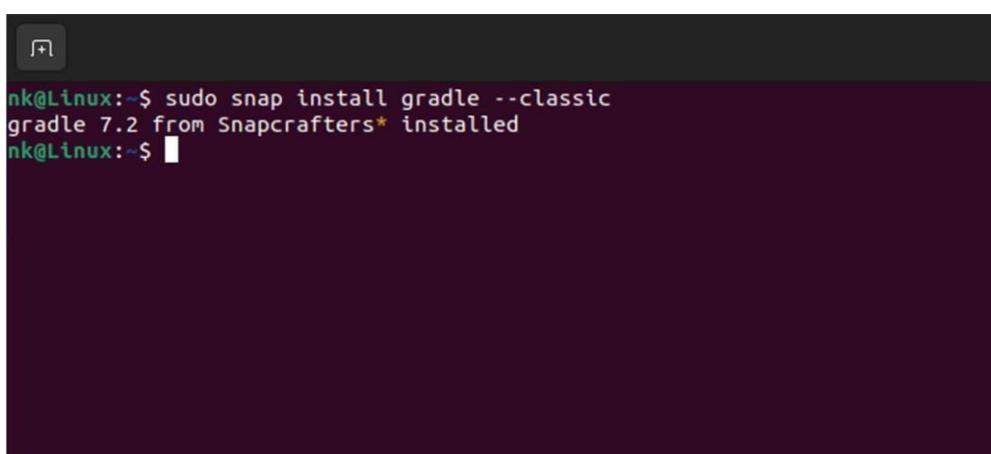
Step 6: Select build script language (Groovy).

Step 7: Choose testing framework (JUnit 4).

Step 8: Enter project name.

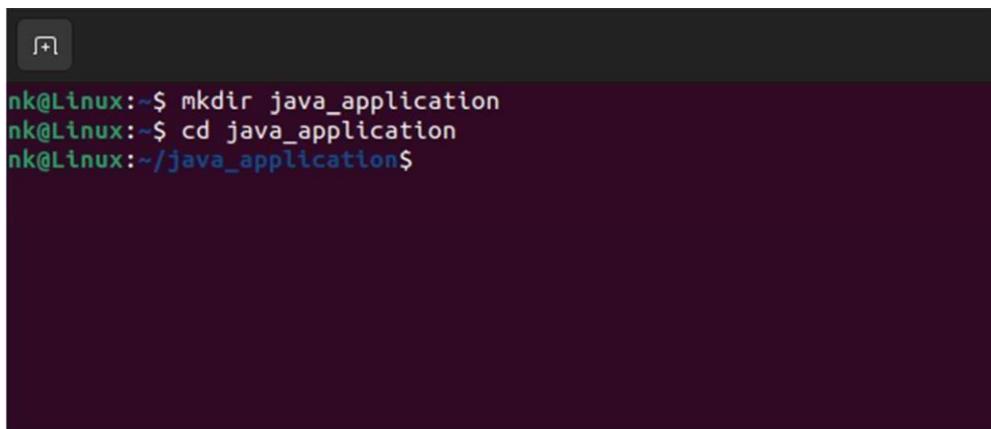
Step 9: Enter source package.

1. Install gradle using the following command



```
nk@Linux:~$ sudo snap install gradle --classic
gradle 7.2 from Snapcrafters* installed
nk@Linux:~$ █
```

2. Create a directory named **java_application** and change directory:



```
nk@Linux:~$ mkdir java_application
nk@Linux:~$ cd java_application
nk@Linux:~/java_application$
```

3. Run the **gradle init** command to create a new Gradle project

```
nk@Linux:~/java_application$ gradle init
Welcome to Gradle 7.2!

Here are the highlights of this release:
- Toolchain support for Scala
- More cache hits when Java source files have platform-specific line endings
- More resilient remote HTTP build cache behavior

For more details see https://docs.gradle.org/7.2/release-notes.html

Starting a Gradle Daemon (subsequent builds will be faster)

Select type of project to generate:
 1: basic
 2: application
 3: library
 4: Gradle plugin
Enter selection (default: basic) [1..4] ■
```

4. Select the project type (application, library, etc.) by typing `2` and pressing Enter for an application.

```
Select type of project to generate:
 1: basic
 2: application
 3: library
 4: Gradle plugin
Enter selection (default: basic) [1..4] 2
```

5. Choose the implementation language (Java, Kotlin, etc.) by typing `3` and pressing Enter for Java.

```
Select implementation language:
 1: C++
 2: Groovy
 3: Java
 4: Kotlin
 5: Scala
 6: Swift
Enter selection (default: Java) [1..6] 3
```

6. Select the default build script language (Groovy or Kotlin) by typing `1` and pressing Enter for Groovy.

```
Split functionality across multiple subprojects?:  
 1: no - only one application project  
 2: yes - application and library projects  
Enter selection (default: no - only one application project) [1..2] 1  
  
Select build script DSL:  
 1: Groovy  
 2: Kotlin  
Enter selection (default: Groovy) [1..2]
```

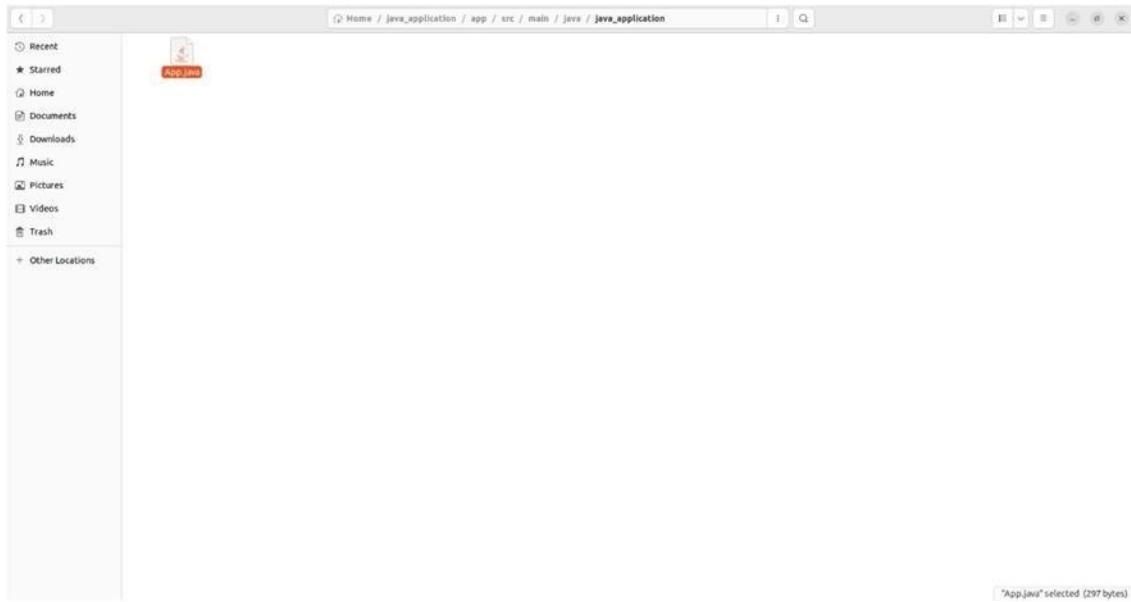
7. Choose the testing framework (JUnit 4, TestNG, etc.) if prompted. The default is Junit4

```
Select test framework:  
 1: JUnit 4  
 2: TestNG  
 3: Spock  
 4: JUnit Jupiter  
Enter selection (default: JUnit Jupiter) [1..4]
```

8. Enter the project name when prompted (default is the directory name) & Enter the source package when prompted (default is the directory name).

```
- More resilient remote HTTP build cache behavior  
For more details see https://docs.gradle.org/7.2/release-notes.html  
Starting a Gradle Daemon (subsequent builds will be faster)  
  
Select type of project to generate:  
 1: basic  
 2: application  
 3: library  
 4: Gradle plugin  
Enter selection (default: basic) [1..4] 2  
  
Select implementation language:  
 1: C++  
 2: Groovy  
 3: Java  
 4: Kotlin  
 5: Scala  
 6: Swift  
Enter selection (default: Java) [1..6] 3  
  
Split functionality across multiple subprojects?:  
 1: no - only one application project  
 2: yes - application and library projects  
Enter selection (default: no - only one application project) [1..2] 1  
  
Select build script DSL:  
 1: Groovy  
 2: Kotlin  
Enter selection (default: Groovy) [1..2] 1  
  
Select test framework:  
 1: JUnit 4  
 2: TestNG  
 3: Spock  
 4: JUnit Jupiter  
Enter selection (default: JUnit Jupiter) [1..4]  
  
Project name (default: java_application):  
Source package (default: java_application):  
  
> Task :init  
Get more help with your project: https://docs.gradle.org/7.2/samples/sample\_building\_java\_applications.html  
  
BUILD SUCCESSFUL in 1m 23s  
2 actionable tasks: 2 executed  
nk@Linux:~/java_application$ █
```

OUTPUT

A screenshot of a code editor window titled "App.java" located at "java_application/app/src/main/java/java_application". The code editor displays the following Java code:

```
1 /**
2 * This Java source file was generated by the Gradle 'init' task.
3 */
4 package java_application;
5
6 public class App {
7     public String getGreeting() {
8         return "Hello World!";
9     }
10    public static void main(String[] args) {
11        System.out.println(new App().getGreeting());
12    }
13 }
14 []
```

The status bar at the bottom right shows "Java" and "Ln 14, Col 2".

```
dhineshkumar1729@dhineshkumar1729-VirtualBox:~$ cd java_application/
dhineshkumar1729@dhineshkumar1729-VirtualBox:~/java_application$ ./gradlew run
Downloading https://services.gradle.org/distributions/gradle-7.2-bin.zip
.....10%.....20%.....30%.....40%.....50%.....
60%.....70%.....80%.....90%.....100%
Starting a Gradle Daemon, 1 incompatible Daemon could not be reused, use --status for details

> Task :app:run
Hello World!

BUILD SUCCESSFUL in 37s
2 actionable tasks: 2 executed
dhineshkumar1729@dhineshkumar1729-VirtualBox:~/java_application$ █
```

RESULT

Thus, the Java application was successfully created using Gradle.

**Ex. No: 8 INSTALL ANSIBLE AND CONFIGURE ANSIBLE ROLES AND TO
Date: WRITE PLAYBOOKS**

AIM

To install Ansible, configure Ansible roles and write playbooks.

PROCEDURE

Step 1: Refresh your system's package index so that it is aware of the packages available.

Run the following command,

```
$ sudo apt update
```

```
*** System restart required ***
Last login: Fri Apr 12 13:39:11 2024 from 13.48.4.203
ubuntu@ip-172-31-43-52:~$ ansible --version
Command 'ansible' not found, but can be installed with:
sudo apt install ansible      # version 2.10.7+merged+base+2.10.8+dfsg-1, or
sudo apt install ansible-core # version 2.12.0-1ubuntu0.1
ubuntu@ip-172-31-43-52:~$ sudo apt update
Hit:1 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu jammy InRelease
Hit:2 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:3 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu jammy-backports InRelease
Hit:4 http://security.ubuntu.com/ubuntu jammy-security InRelease
Ign:5 https://pkg.jenkins.io/debian-stable binary/ InRelease
Hit:6 https://pkg.jenkins.io/debian-stable binary/ Release
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
2 packages can be upgraded. Run 'apt list --upgradable' to see them.
```

Step 2: The Ansible package and its dependencies are available in the default package repositories to install the latest Ansible version, add its ppa repository.

```
$ sudo apt upgrade
$ sudo apt install -y software-properties-common
$ sudo add-apt-repository --yes --update ppa:ansible/ansible
```

```
ubuntu@ip-172-31-43-52:~$ sudo apt upgrade
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Calculating upgrade... Done
The following packages have been kept back:
  python3-update-manager update-manager-core
0 upgraded, 0 newly installed, 0 to remove and 2 not upgraded.
ubuntu@ip-172-31-43-52:~$ sudo apt install software-properties-common
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
software-properties-common is already the newest version (0.99.22.9).
0 upgraded, 0 newly installed, 0 to remove and 2 not upgraded.
```

```

ubuntu@ip-172-31-43-52:~$ sudo apt-add-repository --yes --update ppa:ansible/ansible
Repository: 'deb https://ppa.launchpadcontent.net/ansible/ubuntu/ jammy main'
Description:
Ansible is a radically simple IT automation platform that makes your applications and systems easier to deploy. Avoid writing scripts or custom code to deploy and update your applications— automate in a language that approaches plain English, using SSH, with no agents to install on remote systems.

http://ansible.com/

If you face any issues while installing Ansible PPA, file an issue here:
https://github.com/ansible-community/ppa/issues
More info: https://launchpad.net/~ansible/+archive/ubuntu/ansible
Adding repository.
Adding deb entry to /etc/apt/sources.list.d/ansible-ubuntu-ansible-jammy.list
Adding disabled deb-src entry to /etc/apt/sources.list.d/ansible-ubuntu-ansible-jammy.list
Adding key to /etc/apt/trusted.gpg.d/ansible-ubuntu-ansible.gpg with fingerprint 6125E2A8C77F2818FB7BD15B93C4A3FD7BB9C367
Hit:1 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu jammy InRelease
Hit:2 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:3 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu jammy-backports InRelease
Ign:4 https://pkg.jenkins.io/debian-stable binary/ InRelease
Hit:5 https://pkg.jenkins.io/debian-stable binary/ Release
Hit:6 http://security.ubuntu.com/ubuntu jammy-security InRelease
Get:7 https://ppa.launchpadcontent.net/ansible/ubuntu jammy InRelease [18.0 kB]
Get:9 https://ppa.launchpadcontent.net/ansible/ubuntu jammy/main amd64 Packages [1128 B]
Get:10 https://ppa.launchpadcontent.net/ansible/ubuntu jammy/main Translation-en [752 B]
Fetched 19.9 kB in 1s (19.8 kB/s)
Reading package lists... Done

```

Step 3: Now install the ansible latest version using the command,

```
$ sudo apt install ansible
```

```

ubuntu@ip-172-31-43-52:~$ sudo apt install ansible
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
ansible is already the newest version (9.4.0-1ppa~jammy).
0 upgraded, 0 newly installed, 0 to remove and 2 not upgraded.
ubuntu@ip-172-31-43-52:~$ █

```

Now we are going to navigate the directory path to,

```
$ cd /etc/ansible/
```

Then, we will check the latest version of the ansible by,

```
$ ansible --version
```

```

ubuntu@ip-172-31-43-52:~$ cd /etc/ansible/
ubuntu@ip-172-31-43-52:/etc/ansible$ ansible --version
ansible [core 2.16.5]
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['~/home/ubuntu/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  ansible collection location = ~/home/ubuntu/.ansible/collections:/usr/share/ansible/collections
  executable location = /usr/bin/ansible
  python version = 3.10.12 (main, Nov 20 2023, 15:14:05) [GCC 11.4.0] (/usr/bin/python3)
  jinja version = 3.0.3
  libyaml = True
ubuntu@ip-172-31-43-52:/etc/ansible$ █

```

Step 4: To check the list of directories and files present in the ansible, run the command as,

```
$ ls -la
```

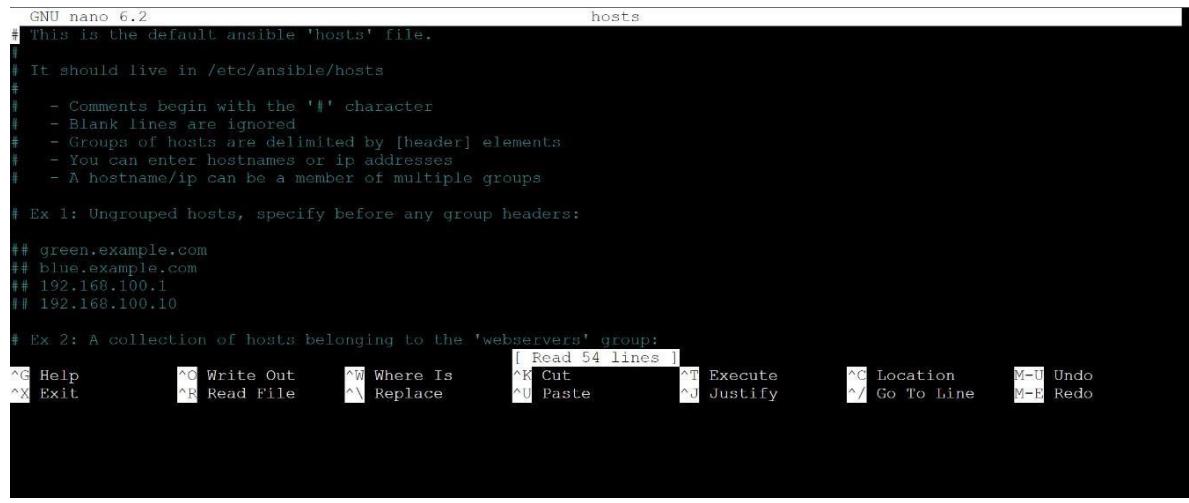
```

ubuntu@ip-172-31-43-52:/etc/ansible$ ls -la
total 20
drwxr-xr-x  3 root root 4096 Apr 12 13:51 .
drwxr-xr-x 103 root root 4096 Apr 12 13:51 ..
-rw-r--r--  1 root root  614 Mar 26 17:11 ansible.cfg
-rw-r--r--  1 root root 1175 Mar 26 17:11 hosts
drwxr-xr-x  2 root root 4096 Mar 26 17:11 roles
ubuntu@ip-172-31-43-52:/etc/ansible$ █

```

Then check the host file by,

```
$ sudo nano hosts
```



The screenshot shows the nano text editor displaying the contents of the /etc/ansible/hosts file. The file contains comments explaining the syntax, examples of ungrouped hosts, and hosts belonging to a 'webservers' group. The nano interface includes standard keyboard shortcuts for operations like Help, Write Out, Cut, Paste, and Undo.

```
GNU nano 6.2                               hosts
# This is the default ansible 'hosts' file.

# It should live in /etc/ansible/hosts

# - Comments begin with the '#' character
# - Blank lines are ignored
# - Groups of hosts are delimited by [header] elements
# - You can enter hostnames or ip addresses
# - A hostname/ip can be a member of multiple groups

# Ex 1: Ungrouped hosts, specify before any group headers:

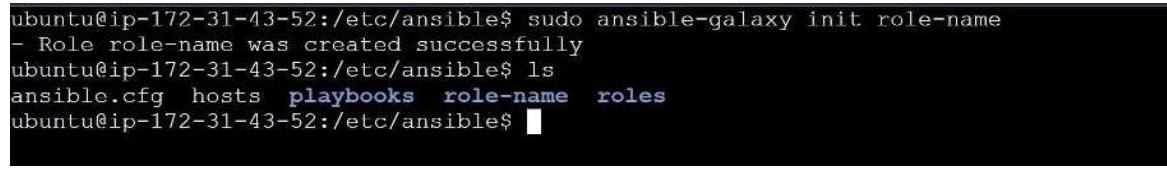
## green.example.com
## blue.example.com
## 192.168.100.1
## 192.168.100.10

# Ex 2: A collection of hosts belonging to the 'webservers' group:
[ Read 54 lines ]
^G Help      ^C Write Out    ^W Where Is      ^K Cut        ^T Execute      ^C Location     M-U Undo
^X Exit      ^R Read File    ^\ Replace       ^U Paste       ^J Justify      ^Y Go To Line   M-E Redo
```

Step 5: Create an ansible role from scratch and run ansible-galaxy command.

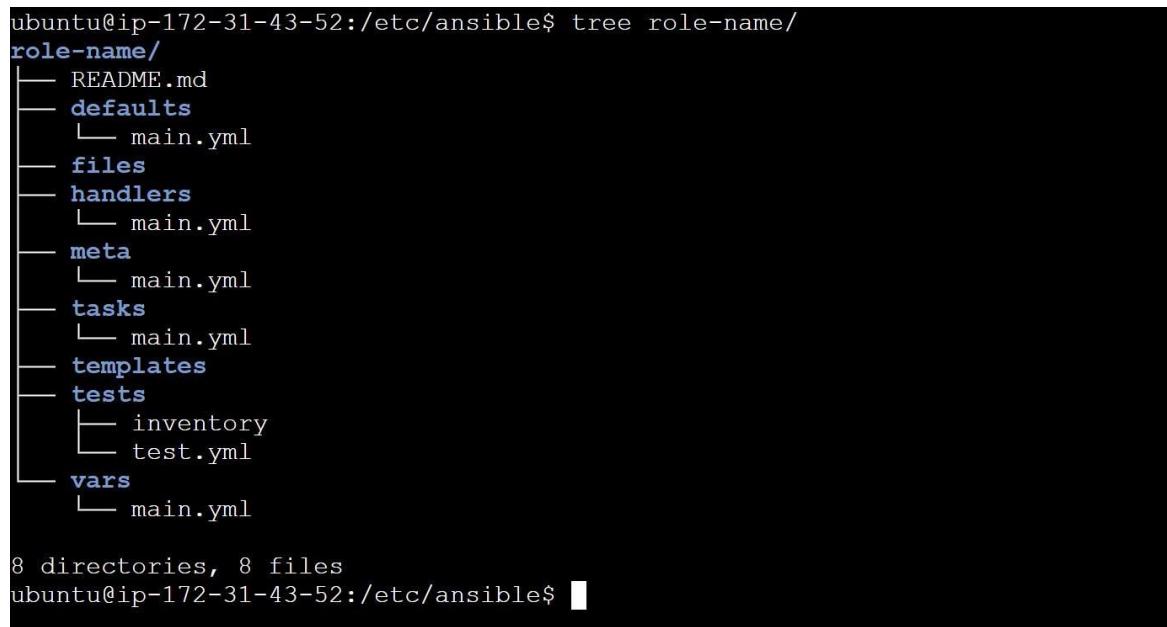
```
$ ansible-galaxy init my-role
```

```
$ ls
```



```
ubuntu@ip-172-31-43-52:/etc/ansible$ sudo ansible-galaxy init role-name
- Role role-name was created successfully
ubuntu@ip-172-31-43-52:/etc/ansible$ ls
ansible.cfg  hosts  playbooks  role-name  roles
ubuntu@ip-172-31-43-52:/etc/ansible$
```

To view the role directory structure, run the tree command followed by the role name.



```
ubuntu@ip-172-31-43-52:/etc/ansible$ tree role-name/
role-name/
├── README.md
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── tasks
│   └── main.yml
├── templates
└── tests
    ├── inventory
    └── test.yml
└── vars
    └── main.yml

8 directories, 8 files
ubuntu@ip-172-31-43-52:/etc/ansible$
```

Step 6: Create three ansible roles as follows:

The prerequisites role – Installs git.

The postgresql role – Installs postgresql service.

The apache role – Installs the Apache webserver.

Use the following commands to create the above,

```
$ sudo ansible-galaxy init prerequisites
```

```
$ sudo ansible-galaxy init PostgreSQL
```

```
$ sudo ansible-galaxy init apache
```

```
$ ls
```

```
ubuntu@ip-172-31-43-52:/etc/ansible$ sudo ansible-galaxy init prerequisites
- Role prerequisites was created successfully
ubuntu@ip-172-31-43-52:/etc/ansible$ sudo ansible-galaxy init PostgreSQL
- Role PostgreSQL was created successfully
ubuntu@ip-172-31-43-52:/etc/ansible$ sudo ansible-galaxy init apache
- Role apache was created successfully
ubuntu@ip-172-31-43-52:/etc/ansible$ ls
PostgreSQL  ansible.cfg  apache  hosts  playbooks  prerequisites  role-name  roles
ubuntu@ip-172-31-43-52:/etc/ansible$ █
```

Step 7: Define each role that you have created. To achieve this, you need to edit the main.yml file located in the ‘tasks’ folder for each role.

First for prerequisites,

```
$ cd prerequisites/tasks/
```

Then edit the main.yml file by the command,

```
$ sudo nano main.yml
```

```
ubuntu@ip-172-31-43-52:/etc/ansible$ cd prerequisites/tasks/
ubuntu@ip-172-31-43-52:/etc/ansible/prerequisites/tasks$ sudo nano main.yml
ubuntu@ip-172-31-43-52:/etc/ansible/prerequisites/tasks$ █
```

Then add the content to this main.yml file as

```
# tasks file for prerequisites
```

```
- name: Install git
```

```
apt:
```

```
  name: git
```

```
  state: present
```

```
  update_cache: yes
```

```

GNU nano 6.2                               main.yml
---
# tasks file for prerequisites
- name: Install git
  apt:
    name: git
    state: present
    update_cache: yes

| Wrote 7 lines |
^G Help          ^O Write Out      ^W Where Is      ^K Cut           ^T Execute       ^C Location      M-U Undo
^X Exit          ^R Read File       ^\ Replace       ^U Paste         ^J Justify       ^/ Go To Line    M-E Redo

```

Next for PostgreSQL,

```
$ cd PostgreSQL/tasks/
```

```
$ sudo nano main.yml
```

Then add the content to this main.yml file as,

```
# tasks file for PostgreSQL
- name: Install PostgreSQL
  apt:
    name: postgresql
    state: present
    update_cache: yes
- name: Start PostgreSQL service
  systemd:
    name: postgresql
    state: started
    enabled: yes
```

```

GNU nano 6.2                               main.yml *
---
# tasks file for PostgreSQL
- name: Install PostgreSQL
  apt:
    name: postgresql
    state: present
    update_cache: yes
- name: Start PostgreSQL service
  systemd:
    name: postgresql
    state: started
    enabled: yes

| Wrote 1 line |
^G Help          ^O Write Out      ^W Where Is      ^K Cut           ^T Execute       ^C Location      M-U Undo
^X Exit          ^R Read File       ^\ Replace       ^U Paste         ^J Justify       ^/ Go To Line    M-E Redo

```

Finally for the apache web server,

```
$ cd apache/tasks/
```

```
$ sudo nano main.yml
```

Then add the content to this main.yml file as,

```
# tasks file for apache
- name: install Apache web server
  apt:
    name: apache2
    state: present
    update_cache: yes
```

The screenshot shows a terminal window with the nano 6.2 editor open. The file is named 'main.yml'. The content of the file is:

```
GNU nano 6.2                               main.yml
# tasks file for apache
- name: install Apache web server
  apt:
    name: apache2
    state: present
    update_cache: yes
```

The terminal window includes a menu bar at the top with 'File', 'Edit', 'Insert', 'Search', 'Format', 'View', and 'Help'. At the bottom, there is a status bar with various keyboard shortcuts for file operations like Help (^G), Exit (^X), Write Out (^O), Read File (^R), Where Is (^W), Replace (^V), Cut (^K), Paste (^U), Execute (^T), Justify (^J), Location (^C), Go To Line (^L), Undo (^U), Redo (^Y), and Request (^R).

Lastly, we are going to create a playbook file called stack.yml and call the roles by the command,

```
$ sudo nano stack.yml
```

Then add the content to this main.yml file as

```
- hosts: localhost
```

```
become: true
```

```
roles:
```

- prerequisites
- PostgreSQL
- apache

```

GNU nano 6.2                               stack.yml

---
- hosts: localhost
  become: true
  roles:
    - prerequisites
    - PostgreSQL
    - apache

[ Wrote 7 lines ]
^G Help          ^C Write Out      ^W Where Is      ^K Cut           ^T Execute       ^C Location      M-U Undo
^X Exit          ^R Read File      ^N Replace       ^U Paste          ^J Justify       ^Y Go To Line    M-E Redo

```

Step 8: Ensure that our roles are working as expected, and run the ansible playbook.

Run the following command,

```
$ sudo ansible-playbook stack.yml
```

```

ubuntu@ip-172-31-43-52:/etc/ansible$ sudo ansible-playbook stack.yml
[WARNING]: provided hosts list is empty, only localhost is available. Note that the implicit localhost does not match 'all'

PLAY [localhost] ****
TASK [Gathering Facts] ****
ok: [localhost]
TASK [prerequisites : Install git] ****
ok: [localhost]
TASK [PostgreSQL : Install PostgreSQL] ****
changed: [localhost]
TASK [PostgreSQL : Start PostgreSQL service] ****
ok: [localhost]
TASK [apache : install Apache web server] ****
changed: [localhost]
PLAY RECAP ****
localhost                  : ok=5    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
ubuntu@ip-172-31-43-52:/etc/ansible$ █

```

Then check the versions to ensure that the packages were installed successfully,

```
$ git --version
```

```
$ apachectl -v
```

```

ubuntu@ip-172-31-43-52:/etc/ansible$ apachectl -v
Server version: Apache/2.4.52 (Ubuntu)
Server built:   2024-04-10T17:45:18
ubuntu@ip-172-31-43-52:/etc/ansible$ git --version
git version 2.34.1
ubuntu@ip-172-31-43-52:/etc/ansible$ █

```

RESULT

Thus, installing ansible, configuring ansible roles and writing playbooks have been executed successfully.